НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

«До захисту допущено»
Завідувач кафедри
_____ О.І. Ролік
«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050201 «Системна інженерія»

на тему: «Застосунок для віддалених замовлень в закладах харчування»

Виконала:
студентка IV курсу, групи ІА-51
Кривошеєва Вікторія Олександрівна _____
Керівник:
ст. викл. каф. АУТС Тимофєєва Ю. С.

_____
Рецензент:

_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2019 рік

Пояснювальна записка
до дипломного проекту
на тему: «Застосунок для віддалених замовлень в закладах харчування»

Київ – 2019 рік

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 14 рисунків, 1 таблицю, 1 додаток, 20 джерел.

Дипломний проект присвячений розробці застосунку для віддалених замовлення в закладах харчуМзвання.

Метою роботи є скорочення часу очікування замовлення в закладах харчування за рахунок зменшення часу на пошук відповідного місця, спілкування з персоналом закладів, автоматизації процесів створення та обробки замовлень.

У розділі огляд існуючих рішень було описано існуючі застосунки зі схожою функціональністю та представлено їх переваги і недоліки.

У розділі вибір підходів та технологій для розробки описано основні засоби розробки застосунку і визначено вимоги до технічного забезпечення.

Розробка застосунку описує архітектуру та реалізацію програмного забезпечення.

У розділі тестування програмного забезпечення проведено випробування програмного продукту.

ABSTRACT

Structure and scope of work. The thesis consists of five sections, contains 14 drawings, 1 table, 1 application, 20 sources.

The thesis is devoted to the development of an application for remote orders in catering establishments.

The aim of the work is to shorten the waiting time for orders in catering establishments by reducing the time to find the appropriate place, communicating with the staff, automating the processes of creating and processing orders.

An overview of existing solutions section describes existing applications with similar functionality and presents their advantages and disadvantages.

The section on choosing product approaches and technologies describes the main tools for developing applications and defines the requirements for technical support.

The development of the application describes the architecture and implementation of the software.

The Software Testing section provides software product testing.

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ У РОБОТІ СКОРОЧЕНЬ

API — Application Programming Interface — сукупність засобів та правил, що вможливлюють взаємодію між окремими складниками програмного забезпечення

AJAX — Asynchronous JavaScript and XML — асинхронний JavaScript та XML

CSS — Cascading Style Sheets — каскадні таблиці стилів

JS — JavaScript — мова програмування

JSON — JavaScript Object Notation — нотація об'єкта javascript

REST — Representational State Transfer, підхід до архітектури мережевих протоколів

HTTP — HyperText Transfer Protocol — протокол передачі гіпертексту

MPA — Multiple Page Application — багатосторінковий веб-застосунок

SPA — Single Page Application — односторінковий веб-застосунок

SEO — search engine optimization — пошукова оптимізація

UX — user experience design — дизайн досвіду користувача

БД — база даних

ПЗ — програмне забезпечення

СУБД — система управління базами даних

ВСТУП

В умовах ринкової економіки наряду з харчовою промисловістю і торгівлею виникла і успішно розвивається така галузь господарства, як громадське харчування, що замінює домашнє приготування їжі суспільним виробництвом з застосуванням сучасних технологій та обладнання.

Громадське харчування, яке виникло як результат громадського розподілу праці, сприяє ощадливій витраті праці і створює умови для прийому їжі поблизу місць роботи, навчання і відпочинку. Досить часто темп життя сучасної людини не дозволяє приділяти достатньо уваги своєму раціону, режиму харчування, і тоді вона вдається до вживання доступних фастфудів, продуктів швидкого приготування. Витрачаючи більшість часу на кар'єру, люди забувають про найдорожчі речі, які не можна отримати через матеріальні цінності, а саме про втрачений час та здоров'я.

Розширення громадського харчування дозволяє в значному ступені збільшити вільний час робітників, що служить одним з важливих шляхів всебічного гармонійного — фізичного і духовного — розвитку людини, призводить до підвищення продуктивності суспільної праці. Такий вид харчування дозволяє відновлювати витрачені сили і забезпечує спроможність робітників до праці.

Наближення мережі підприємств громадського харчування до фабрик і заводів, до навчальних і наукових установ сприяє створенню правильного режиму харчування, а також економить час для відпочинку людям, які працюють і навчаються.

Саме тому, метою даного дипломного проекту є створення застосунку для віддалених замовлень в закладах харчування.

Завдання полягає у створенні веб-застосунку, який дозволить користувачам створювати індивідуальні профілі, робити та оплачувати замовлення онлайн, відстежувати статус замовлення, отримувати рекомендації на основі по-

передніх відгуків та геолокації користувача, а також переглядати рейтинг закладу і залишені іншими відвідувачами відгуки. Застосунок дозволить працівникам закладу швидше приймати замовлення та обслуговувати відвідувачів не витрачаючи час на спілкування з клієнтами. Миттєва оплата замовлення забезпечить заклад від зайвих витрат часу та грошей, якщо користувач передумає забирати замовлення.

Як показує практичний досвід, додатки для ресторанів і кафе значно збільшують середні чеки і оптимізують робочі процеси, знижують витрати на обслуговування персоналу, збільшують клієнтуру. Смартфони середньостатистичних користувачів завжди мають доступ до інтернету. Ні для кого не складе труднощів перейти на сайт сервісу. При цьому у виграші від впровадження такої автоматизації будуть як користувачі, так і власники закладу.

Оскільки сучасні технології дозволяють створювати кросплатформенні додатки, тому орієнтування на конкретну платформу немає. Система повинна мати можливість розгортання на всіх платформах, таких як Windows, Mac, Linux, а також на мобільних пристроях.

Таким чином, автоматизація діяльності закладів харчування є необхідним і перспективним процесом. Адже комплексне використання сучасних інформаційних технологій дозволяє значно облегшити роботу закладів і суттєво зекономити час відвідувачам.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Веб-додатки мимоволі замінюють старі програми для персональних комп’ютерів столу. Вони більш зручні у використанні, їх легко оновлювати, і вони не прив’язані до одного пристрою. І навіть якщо користувачі злегка переходять від веб-додатків на основі браузера до мобільних, попит на складні та вдосконалені програми величезний і продовжує зростати.

У зв'язку з бурхливим розвитком інформаційних технологій, кількість адаптивних веб-сайтів все більше зростає.

Адаптивний або мобільний веб-сайт подібний до будь-якого іншого - він використовує HTML-сторінки, до яких можна отримати доступ з будь-якого браузера. Так як звичайний веб-сайт створений для перегляду на персональному комп’ютері, адаптивний сайт, перш за все, спеціально розроблений для показу на невеликих екранах, таких як мобільні телефони та планшети.

Адаптивний сайт має спеціальний дизайн, заснований на медіа запитах CSS. Це дозволяє вмісту веб-сайту вписуватися в будь-який тип екрана як 17-дюймового ноутбука, так і маленького екрана телефону. Тим не менше, ці запити мають незначні недоліки: вони не тільки потребують більшої роботи та уваги розробника, але й, швидше за все, такий сайт буде завантажуватися повільніше.

Кількість таких веб-сайтів зростає зі збільшенням кількості мобільних телефонів. Так за останні 5 років, кількість користувачів мобільних телефонів в усьому світі збільшилась з 4,01 у 2013 році до 5,07 мільярда у 2019 році. У 2018 році майже 63% населення Землі вже володіли мобільним телефоном. Протягом наступних років ця сума зросте, зокрема, завдяки Китаю та Індії. За прогнозами, у 2022 році кількість виросте до 5.5 мільярдів. На рисунку 1.1 приведена статистика кількості користувачів мобільних телефонів з 2013 по 2019 роки.

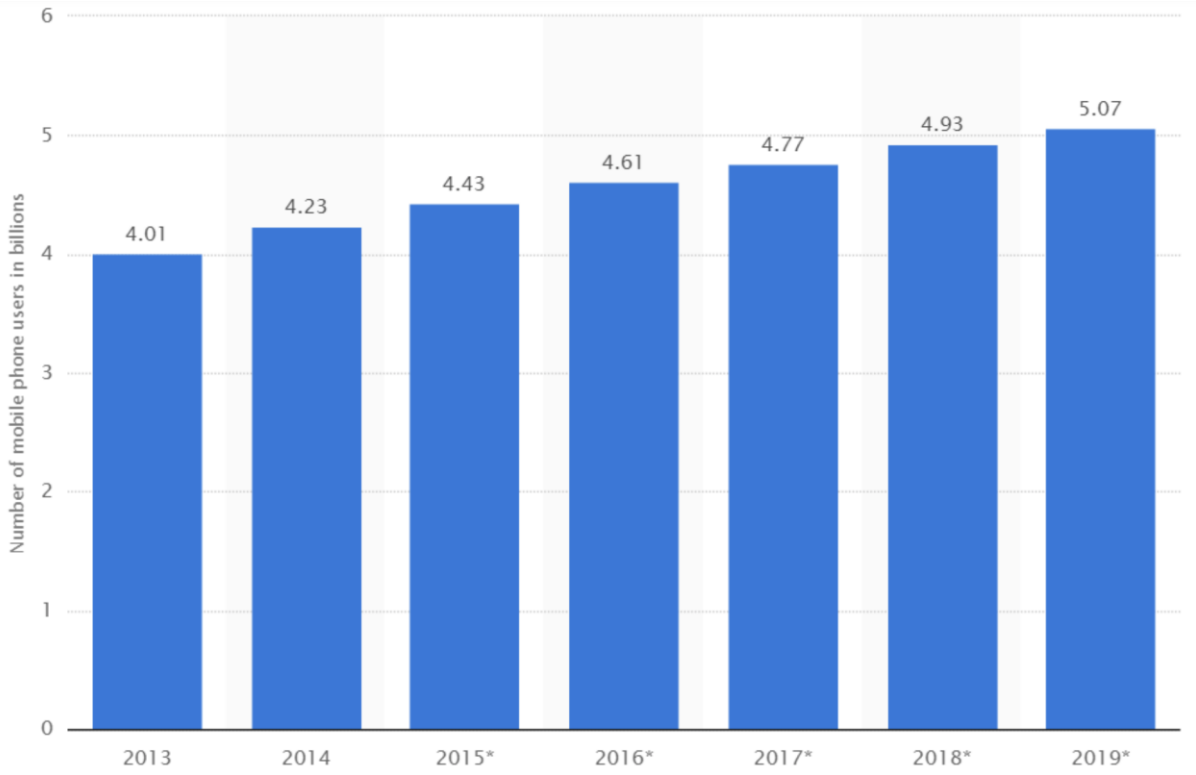

Рисунок 1.1 — Статистика кількості користувачів мобільних телефонів

Багато компаній сьогодні мають адаптивні веб-сайти, щоб зробити свою інформацію, товари або послуги доступними на різних платформах і пристроях. Це особливо важливо, для залучення нових клієнтів.

Веб сайти мають багато переваг. Коли люди шукають інформацію, вони не відразу переходять до магазинів з програмами для мобільних телефонів. Перш за все люди починають пошук з веб-браузерів. Результати таких пошуків складаються в основному з веб-сторінок і лише декількох додатків. Якщо ж шукати в магазинах додатків, шанс знайти потрібну інформацію серед тисяч подібних застосунків дуже маленький.

Адаптивні веб-сайти доступні для всіх користувачів. Їх не потрібно завантажувати та встановлювати. Більшість сайтів — безкоштовні. Користувачі можуть використовувати веб-сайти не залежно від операційної системи як на персональному комп’ютері, так і на мобільних пристроях.

При роботі з сайтами їх легко оновлювати, підтримувати та виправляти помилки. Користувачеві не потрібно встановлювати нову версію, щоб переглянути покращення на веб-сайті. Більшість користувачів навіть не помічають процес оновлення. Вони можуть просто насолоджуватися оновленим продуктом. У той же час, власникові простіше і дешевше оновити один веб-сайт.

Адаптивний веб-сайт дешевше. Коли є один адаптивний веб-сайт, немає потреби створювати нативні програми для кожного типу мобільних телефонів і планшетів. Це також знижує вартість обслуговування та полегшує роботу з SEO.

1.1 Аналіз типів веб-застосунків

Існують дві основні моделі розробки веб-додатків: багатосторінковий веб-застосунок (MPA) і односторінковий застосунок (SPA). Кожна з моделей має свої плюси і мінуси.

SPA - це програма, яка працює в браузері й не вимагає перезавантаження сторінок під час використання (рисунок 1.2).

Цей тип застосунків дуже поширений. Серед відомих односторінкових застосунків, які використовуються щодня: Gmail, карти Google, Facebook, GitHub. SPA призначені для представлення видатного UX, намагаючись наслідувати «природне» середовище в браузері — жодна сторінка не перезавантажується, немає додаткового часу очікування. Це лише одна веб-сторінка, яку користувач відвідує, а потім завантажуєте весь інший вміст сторінки, використовуючи JavaScript.

SPA самостійно запитує розмітку і дані і надає сторінки прямо в браузері. Це можливо завдяки розширеним фреймворкам JavaScript, таким як AngularJS, Ember.js, Meteor.js, Knockout.js. Односторінкові сайти допомагають утримати користувача в одному, зручному веб-просторі, де вміст подається користувачеві простим, легким і зручним способом.

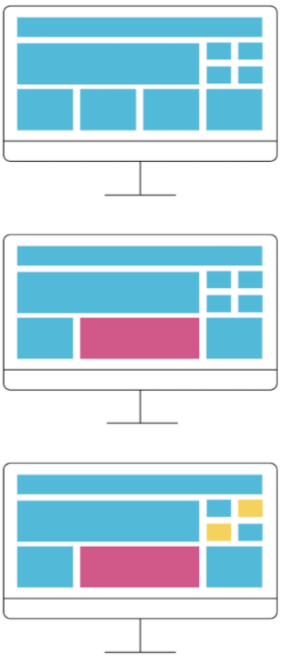

Рисунок 1.2 — Відображення даних в односторінковому веб-застосунку

Переваги односторінкових веб-застосунків:

– SPA є швидким, оскільки більшість ресурсів (HTML + CSS + Scripts) завантажуються лише один раз протягом всього життя програми. Тільки дані передаються вперед і назад;

– Процес розробки спрощений. Немає необхідності писати код для візуалізації сторінок на сервері. Почати розробку набагато простіше, тому що є можливість запустити розробку з файлу, не використовуючи жодного сервера;

– SPA легко піддаються налагодженню за допомогою Chrome, оскільки можна контролювати мережеві операції, досліджувати елементи сторінки та дані, пов'язані з нею;

– Легше зробити мобільний додаток, оскільки розробник може повторно використовувати той же код для веб-застосунків і нативного мобільного додатка;

– SPA може ефективно кешувати будь-яке локальне сховище. Додаток надсилає тільки один запит, зберігає всі дані, потім може використовувати ці дані і працювати навіть в автономному режимі.

– Недоліки односторінкових веб-застосунків:

– Дуже складно і непросто зробити SEO оптимізацію односторінкового веб-застосунку. Його вміст завантажується за допомогою AJAX (Asynchronous JavaScript and XML) - метод обміну даними і оновлення в додатку без оновлення сторінки;

– Завантаження відбувається повільно, оскільки для завантаження клієнта потрібні важкі клієнтські фреймворки;

– Для роботи застосунку потрібен JavaScript, щоб він був присутній і включений. Якщо будь-який користувач вимикає JavaScript у своєму веб-переглядачі, додаток не буде працювати правильно;

– У порівнянні з «традиційним» застосунками, SPA є менш захищеним. Завдяки крос-сайт сценаріям (XSS), зловмисникам можуть додавати клієнтські сценарії в веб-додаток інших користувачів;

– Витік пам'яті в JavaScript може навіть викликати уповільнення потужної системи.

Багатосторінкові програми (MPA) працюють у «традиційний» спосіб. Кожна зміна, така як відображення даних або надсилання даних на сервер створює запити, що видають нову сторінку з сервера в браузері. Це представлено на рисунку 1.3.

Такі застосунки більші ніж SPA. Завдяки кількості контенту ці програми мають багато рівнів інтерфейсу користувача. Проте це вже не проблема. Завдяки AJAX не потрібно турбуватися про те, що великі і складні програми повинні передавати багато даних між сервером і браузером. Таке рішення покращується і дозволяє оновлювати лише окремі частини програми. З іншого боку, це додає більшої складності і такі додатки складніше розробляти ніж односторінкові застосунки.

До переваг MPA можна віднести дуже хороший і легкий спосіб управління SEO. Це дає кращі шанси на ранжування для різних ключових слів, оскільки програма може бути оптимізована для одного ключового слова на сторінці. Також це ідеальний підхід для користувачів, яким потрібна візуальна мапа місця, куди переходити в програмі. Навігація меню з декількох рівнів є важливою частиною традиційного багатосторінкового веб-застосунку.

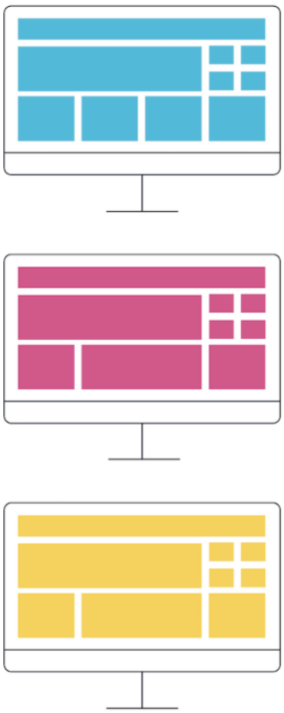

Рисунок 1.3 — Відображення даних в багатосторінковому веб-застосунку

Недоліки багатосторінкових веб-застосунків:

– Фронтенд і розробка бекенда тісно пов’язані;

– Розробка є досить складною. Розробник повинен використовувати фреймворки для клієнтської та серверної частин. Це призводить до більш тривалого часу розробки подібних додатків.

Гібридний додаток приймає те, що є найкращим в обох підходах, і намагається звести до мінімуму недоліки. Це додаток з однією сторінкою, який використовує якорі URL як синтетичні сторінки, що дозволяють створити більше

функціональних переваг. Багато програм на ринку переходять на цю модель. Скоріш за все, у майбутньому кожен користуватиметься моделлю Single Page Application (включаючи гібридний додаток), оскільки така модель приносить багато переваг. Однак, оскільки деякі проекти просто не можуть вписатися в SPA, модель MPA все ще залишається живою.

Отже, після аналізу переваг та недоліків кожного з видів веб-застосунків, для досягнення мети даного дипломного проекту була вибрана гібридна модель.

1.2 Опис функціональної моделі

Специфікацію функціональної поведінки системи представимо у вигляді діаграми варіантів використання. На рисунку 1.4 наведено всі функції системи та описано акторів, які будуть їх використовувати.

Рисунок 1.4 – USE-case діаграма

Безпосередньо із системою будуть взаємодіяти три актори: клієнт, адміністратор та офіціант. Відповідно до визначених варіантів використання визначено функціональні вимоги. Результат для актора «клієнт» наведено в таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги для клієнта

Варіант використання	**Функціональна вимога**
Реєстрація	Для створення облікового запису потрібно натиснути кнопку «Реєстрація», щоб увійти на сторінку створення облікового запису. У відповідні поля ввести користувацькі дані. Якщо реєстрація відбулась успішно, користувач буде перенаправлений на головну сторінку. В іншому випадку з’явиться повідомлення про помилку.
Вхід	Після реєстрації клієнт може увійти до сайту з правильним ім'ям користувача та паролем.
Перегляд списку закладів	На головній сторінці користувач має можливість переглянути список закладів і відсортувати її за категоріями.
Перегляд рекомендованих закладів	Натиснувши на кнопку «Мапа» користувач має можливість переглянути рекомендовані заклади найближчі по геолокації.
Перегляд меню	Вибравши заклад у списку закладів, користувач має можливість переглянути

	список страв відповідно до категорій. Натиснувши кнопку «Детальніше», можна переглянути більш детальну інформацію про страву.
Створення замовлення	Натиснувши кнопку «Замовити» користувач формує нове замовлення.
Здійснення оплати	Після перевірки правильності замовлення, користувач натискає «Оплатити» та переходить на сторінку здійснення оплати.
Перегляд статусу замовлення	Після створення замовлення користувач може переглядати статус виконання замовлення в реальному часі.
Створення відгуку	Після завершення замовлення, користувач може залишити відгук та поставити оцінку закладу натиснувши кнопку «Оцінити»
Бронювання столика	Натиснувши кнопку «Забронювати» користувач може забронювати місце в закладі.
Перегляд історії замовлень	Натиснувши кнопку «Історія» користувач може переглянути залишені відгуки та інформацію про закриті замовлення.

1.3 Постановка задачі

1.3.1 Призначення розробки

Запропонований програмний продукт призначений для створення попередніх замовлень в закладах харчування. Завдання даного веб-застосунку є максимально спростити процеси, пов'язані з відвідуванням закладів харчування, таких як пошук відповідного місця, бронювання столика, замовлення страв та оплата рахунку. Застосунок зводить потребу в спілкуванні зі співробітниками закладів до мінімума, що значно економить час відвідувачів.

1.3.2 Цілі та задачі розробки

Основна мета роботи – зекономити час на відвідування закладів громадського харчування за рахунок зменшення часу на пошук відповідного місця, спілкування з персоналом закладів, автоматизації процесів створення та обробки замовлень.

Даний сервіс повинен дозволяти користувачам створювати замовлення в закладах громадського харчування з будь якого місця та з будь-якого пристрою, де є доступ до мережі Інтернет. Працівники закладів повинні мати можливість завантажувати і оновлювати меню, приймати замовлення і обробляти їх. Повинен бути реалізований функціонал для відстежування статусу готовності замовлення. Повинен бути реалізований модуль, який відповідає за зворотній зв’язок клієнтів з робітниками закладів, щоб клієнти мали змогу залишити свій відгук. Також потрібна реалізація функціоналу для створення рекомендацій закладів на основі оцінок користувача.

Для реалізації поставленої мети необхідно розв’язати наступні задачі:

1) Налаштувати взаємодію з базою даних;
2) Створити клієнтську частину з розділенням функціоналу для відвідувачів та персоналу;

3) Створити сервер для налаштування взаємодії користувача з персоналом;

4) Налаштувати авторизацію;

5) Створити можливість оплачувати замовлення за допомогою застосунку.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Розвиток нових інформаційних технологій має велике значення для нас. Нові пристрої активно з'являються у нашому житті і через деякий час стають невід'ємною його частиною. Впровадження інформаційних технологій створює підґрунтя для розвитку нової культури праці і одночасно призводить до стратегічної переорієнтації підприємства. Використання інформаційних комп'ютерних систем для вирішення управлінських та підприємницьких завдань, стратегічного розвитку, підвищення ефективності адміністративної діяльності, обліку і контролю, планування й аналізу, реалізації у мережевому режимі різноманітних зв'язків підприємств з їх партнерами, клієнтами, владними структурами призвело до зростання інформаційних потреб, дало можливість не обмежувати інформаційні процеси межами окремого підприємства, а також зумовило зростання інвестицій у комп'ютерні технології.

Згідно з даними Державної служби статистики в Україні нині працює майже 3200 підприємств, у яких зайнято 215 тис. людей, з них 40 тис. – висококваліфіковані спеціалісти, котрі займаються безпосередньо ІТ технологіями. Окрім цього, статистика налічує близько 2 тис. різних організацій та компаній, які також займаються проблемами комп'ютерної індустрії [1]. Персональний комп'ютер, мобільні пристрої та мережа Інтернет дали змогу створювати загальнодоступну, максимально інформаційну та, порівняно з іншими інформаційно-технологічними системами, дешеву й швидку інфраструктуру, їх доступність та надійність сприяли входженню у всі сфери суспільства нових інформаційних технологій.

Так з'являється все більше гнучких, швидких і зручних систем, які допомагають в роботі закладам харчування. При цьому розвиток інтернет-сервісів, а також популярність надтонких клієнтів (програм, що працюють

через браузер) призводить до того, що кількість онлайн-засобів вже перевищує кількість окремо встановлюваних на комп'ютері програм.

З плином часу та розвитком IT технологій, звичайна людина все більше має можливості економити свій час не витрачаючи його на речі, які можна робити віддалено і з найменшим втручанням інших людей.

Серед найбільш відомих сервісів надання послуг віддалених замовлень в закладах харчування можна виділити: Allset, Eda.UA, Glovo. В даному пункті буду розглянуті лише декілька, найбільш відомі кожний в своєму направлені.

2.1 Allset

Allset - сервіс, що дозволяє робити попередні замовлення у ресторанах, бронювати столики, заздалегідь оплачувати замовлення та залишати чайові. Стартап зосереджений на обслуговуванні ресторанів у великих американських містах, таких як Сан-Франциско, Нью-Йорк, Лос-Анджелес, Чикаго, Х`юстон та ін. Станом на 2019 рік Allset працює з більш ніж 1700 ресторанами в 11 містах США. Серед партнерів такі відомі ресторани та мережі ресторанів як La Mar, by CHLOE., BJ's Restaurant & Brewhouse, Buffalo Wild Wings, та ін.[2]

Сервіс дозволяє скоротити очікування замовлення в ресторані до мінімуму, заздалегідь забронювавши столик, зробивши замовлення та оплативши його. Для ресторанів Allset зручний тим, що допомагає обслуговувати більшу кількість клієнтів та гарантувати своєчасне обслуговування клієнтам що поспішають.

Бізнес-модель стартапу станом на 2019 рік передбачає надання двох видів сервісу:

– для ресторанів - що передбачає власне програмне забезпечення із попередні замовлення їжі, бронювання столика та оплати заздалегідь;

– для бізнесу - можливість як маленьким так і великим компаніям оплачувати чи частково покривати вартість страв у ресторанах для своїх працівників через корпоративну версію Allset.

Сервіс має багато переваг:

– мобільний і веб застосунок;

– корпоративна версія;

Серед недоліків можна виділити:

– відсутність механізму відслідковування статусу замовлення;

– відсутність зворотнього зв’язку між клієнтами і закладами;

– відсутність функціоналу, який би «радив» заклади найближчі до місця знаходження клієнта на основі його вподобань;

– сервіс не працює з закладами України.

–

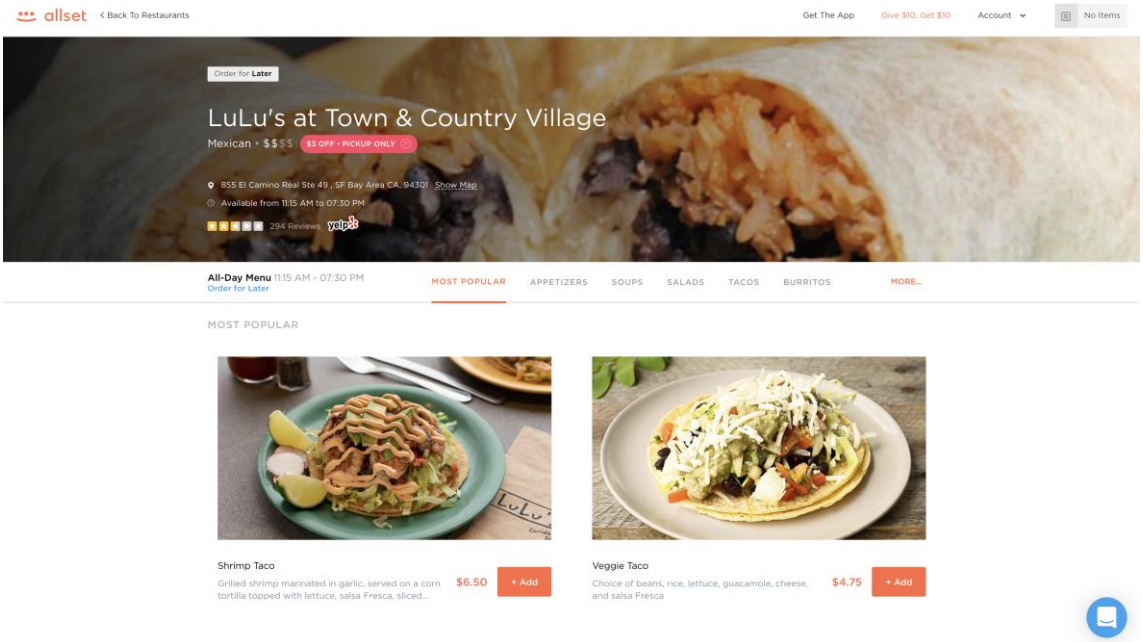

Рисунок 2.1 — Інтерфейс Web-застосунку Allset

2.2 Eliot

Мобільний додаток Eliot - це агрегатор ресторанів, кафе, барів та інших закладів громадського харчування. Його завдання - максимально спро-

стити процеси, пов'язані з відвідуванням таких закладів: від пошуку відповідного місця і бронювання столика до замовлення страв та оплати рахунку. Переваги такого застосунку: більшість дій відбувається в електронному вигляді. Зі співробітниками ресторанів доводиться спілкуватися значно менше, а отже, це сильно економить час.

Додаток повністю безкоштовний. Для початку роботи бажано зареєструватися, але багато функцій працюють і без цього. Наприклад, можна шукати ресторани на карті, читати інформацію про них або вивчати меню. Але для того, щоб прив'язати карту для оплати або збирати бонуси, потрібно авторизуватись.

Авторизація можлива тільки за номером телефону. Інші популярні способи, на зразок ящика електронної пошти або аккаунтів в соцмережах, не підтримуються.

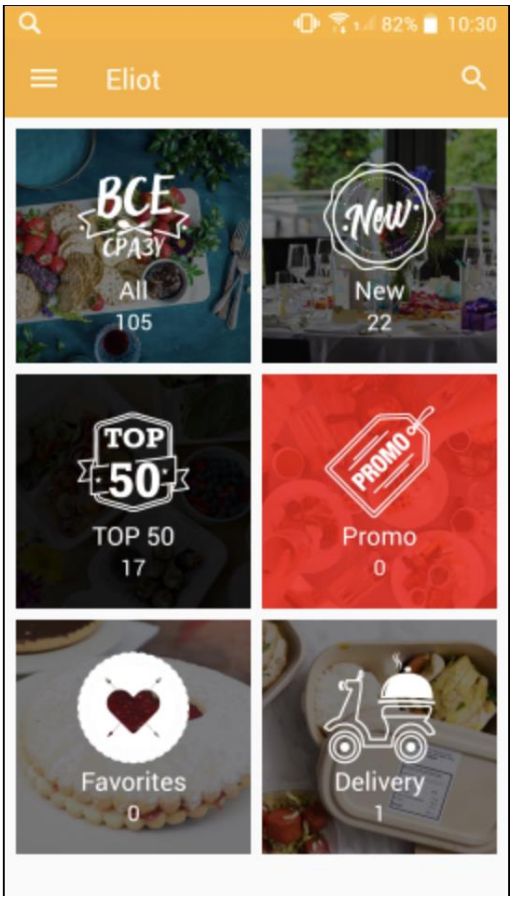

Рисунок 2.2 — Головна сторінка мобільного застосунку Eliot

На стартовому вікні Eliot розташовані шість квадратних плиток, натискання на які відкриває різні розділи програми. Серед розділів є такі: розділ з усіма доступними закладами, ТОП-50, розділ сформований на основі призначених для користувача оцінок, розділ з промо-акціями, заклади з доставкою замовлень і список обраного.

Ділення закладів за типами немає, що ускладнює пошук відповідного місця. Наприклад, якщо користувач хоче пообідати, йому доведеться пошукати підходящі ресторани в загальному списку, серед кав'ярень і барів. Адже за назвою на вивісці не завжди можна зрозуміти, що за нею ховається. Як варіант, можна подивитися розташування найближчих закладів на карті або скористатися пошуком за назвою, якщо вони відомі.

Відкривши картку вибраного місця, користувач бачить адреса закладу, номер телефону, розклад роботи, приблизний цінник на одну людину, опис та відгуки, якщо хтось встиг їх залишити.

Бронюючи столик, відвідувач вказує дату, час і кількість персон. Розробники обіцяють, що в майбутньому можна буде вибирати навіть окремий столик на схемі. Також незабаром користувачі зможуть вносити гроші на депозит, оплачуючи наперед відпочинок в нічних клубах.

Зробити замовлення - просто. Користувач зазначає, чи буде він їсти в ресторані або забере їжу з собою, а потім додає в кошик страви, які і оплачує їх прив'язаною картою. Страви зручно розсортовані по категоріям.

Найближчим часом автори проекту обіцяють додати в Eliot лічильник калорій. В майбутньому програма навчиться на основі уподобань користувача підбирати рекомендовані страви з меню і показувати діючі на них знижки і акції.

Також даний сервіс дозволяє не носити знижку різних ресторанів з собою, а мати вже готову базу всіх акцій і знижок в одному додатку.

Ще одна функція додатку, реалізована в Android-додатку - фільтри для графічної обробки зображень.

Переваги сервісу:

– зручний інтерфейс;

– можливість залишати відгуки;

– можливість замовлень з собою;

– наявність розділу з закладами, сформований на основі призначених для користувача оцінок користувача.

Недоліки:

- багато зайвого функціоналу, який напряму не стосується основних функцій закладу (фільтри для графічних зображень, лічильник калорій);

- тільки мобільний додаток;

- відсутність механізму відслідковування статусу замовлення;

- сервіс не працює з закладами України.

2.3 Skip

Skip - Австралійський сервіс, який допомагає людям робити попередні замовлення зі смартфонів в місцевих кав'ярнях [3].

Користувачі можуть здійснювати пошук у відкритих місцях поблизу, натискати на елементи, які вони хотіли б замовити, налаштовувати їхні замовлення (наприклад, додати додаткові інгредієнти), оплатити кредитною або дебетовою карткою, а потім вибрати час, о котрій отримати замовлення. Skip пропонує інтеграції картки лояльності.

Панель інструментів для постачальників включає в себе функціональні можливості, такі як історія попередніх замовлень, контроль рівня запасів і щоденні підсумки купівлі.

Переваги сервісу:

– дуже простий і зручний інтерфейс;

– можливість замовлень з собою;

– наявність мобільного і веб-застосунку;

– можливість редагувати замовлення.

Недоліки:

– відсутність механізму відслідковування статусу замовлення;

– сервіс не працює з закладами України;

– сервіс призначений тільки для кав’ярень;

– відсутність зворотнього зв’язку між клієнтами і закладами;

– відсутність функціоналу, який би «радив» заклади найближчі до місця знаходження клієнта на основі його вподобань.

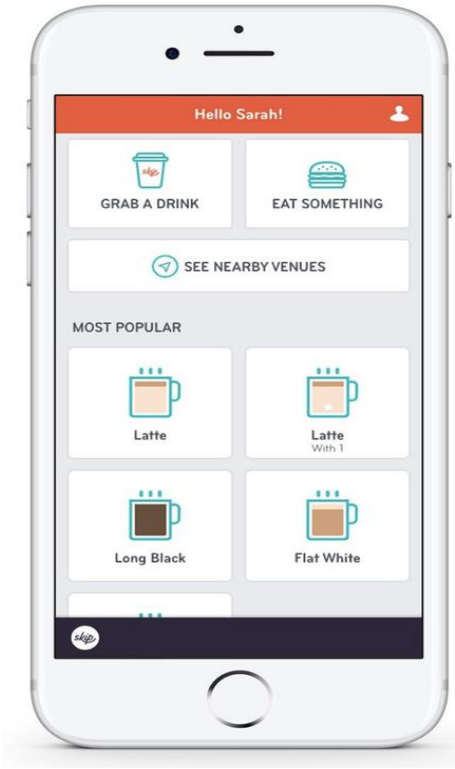

Рисунок 2.3 — Головна сторінка мобільного застосунку Skip

В розділі «Огляд існуючих рішень» було проведено пошук аналогів запропонованого застосунку з подібним функціоналом. Було порівняно функції, які виконують знайдені програмні продукти. Було виявлено декілька систем зі схожим функціоналом, але вони не охоплюють увесь спектр функцій, які запропоновані в дипломному проекті.

3 ВИБІР ПІДХОДІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ

Застосунок розроблений за допомогою використання мови програмування JavaScript. Розробка клієнтської частини включає використання однієї з найпопулярніших бібліотек - React.Js, мови розмітки гіпертексту HTML. Для розробки серверної частини використовується гнучкий фреймворк для веб-застосунків, побудованих на Node.js — Meteor.

3.1 Основні відомості про мову програмування JavaScript

JavaScript є інтерпретованою мовою програмування з об'єктно-орієнтованими можливостями [4]. Поряд з HTML і CSS, JavaScript є однією з трьох основних технологій у веб-розробці з HTML, що описує зміст, CSS, що описує, як відображається вміст, і JavaScript, що описує поведінку контенту. Таким чином, JavaScript здатний працювати на всіх сучасних браузерах без додаткових плагінів або компіляторів і використовується на більшості сучасних веб-сайтів.

JavaScript надає багато хороших можливостей, таких як функції, вільне введення тексту, динамічні об'єкти та нотаріальне позначення об’єкта.

Функції JavaScript - це об'єкти першого класу з лексичним визначенням. Ця мова має більше спільного з Lisp і Scheme, ніж з Java. Це робить JavaScript надзвичайно потужною мовою [5].

Об'єкти вільного друку дозволяють розробникам створювати складні ієрархії класів і зменшують занепокоєння щодо системи типів.

JavaScript має виразну і потужну літеральну нотацію об'єкта. Об'єкти можна створювати, просто перераховуючи їх компоненти. Ця нотація була натхненням для JSON, популярного формату даних.

Оскільки мова JavaScript може бути складною для підтримки всіх різних вимог веб-браузера, існує багато бібліотек JavaScript для полегшення розроб-

ки. Хоча фреймворки JavaScript мають спільні характеристики, такі як маніпулювання DOM, кожен фреймворк має різні основні функціональні можливості. Наприклад, jQuery, одна з найбільш часто використовуваних бібліотек на веб-сайтах, використовує CSS-селектори для керування елементами HTML [6].

3.2 Мова розмітки гіпертексту HTML

HyperText Markup Language, або мова гіпертекстової розмітки сторінок, з'явилася в 1992 році та використовується дотепер [7], але, звісно, у більш вдосконаленому вигляді.

HTML – мова розмітки, що має наступні можливості:

а) представляти документи в мережі у вигляді електронних документів із вмістом у вигляді форматованого тексту, таблиць, списків, фото;

б) включати в документи звукові фрагменти, а саме: відео, електронні таблиці, елементи мультимедіа;

в) завантажувати документи шляхом активізації гіперпосилання;

г) розробляти форми для безпосередньої роботи з віддаленими службами (пошуковими роботами, онлайн магазинами) [8].

HTML має свою особливу розмітку. Вона заключається в тому, щоб представити документ у вигляді послідовності елементів [9].

Документ у форматі .html складається з трьох основних частин:

а) рядки, що оголошують файл як документ, написаний на HTML5;

б) заголовок, що міститься в тезі HEAD;

в) тіло документа, що представлене тегом BODY [10].

3.3 Meteor.JS

Meteor JS - це повноцінний фреймворк з відкритим вихідним кодом, який використовується для розробки веб- і мобільних додатків. Він складається з існуючих технологій, які об'єднуються для створення будівельного блоку програми. Компоненти можуть змінюватися залежно від вимог додатків.

Зростання Meteor було швидким, оскільки він забезпечував підтримку, зокрема, базовою інфраструктурою розвитку, такою як синхронізація даних та компіляція коду. Тому під час розробки можна зосередитися лише на бізнес-функціональності програми. Поточна версія Meteor.JS становить 1,4 і підтримує операційні системи Windows, Mac OS і Linux.

Основною мета Meteor створити базову інфраструктуру, таку синхронізацію даних, яка розгортає додаток на веб-пристроях або мобільних пристроях. Meteor також зосереджується на використанні бібліотек програмного забезпечення, які можна налаштувати з існуючих модулів Node.js для реалізації функціональних можливостей [11]. Інша область полягає в тому, щоб сприяти використанню стандартизованих протоколів, таких як протокол розподілених даних (DDP), і можливість надання послуг розробнику.

Платформа розвитку Meteor складається з технологій, що сприяють процесу розробки програми. Технології - це засоби побудови, набір програмних пакунків, веб-сокети і MongoDB. Процес розробки Meteor.JS використовує JavaScript у всіх середовищах; сервер додатків, веб-браузер і мобільний пристрій, який працює як в інтерфейсі, так і в бекенд, і використовує той же інтерфейс прикладного програмування (API). Комбіновані технології дозволяють розробнику створювати реактивні програми.

На рисунку 3.4 показаний короткий опис основних компонентів, необхідних для запуску програми Meteor. Що стосується сервера, то він складається з MongoDB, Node.js і коду програми. MongoDB забезпечує зберігання даних, а Node.js - середовище JavaScript на стороні сервера, це масштабована се-

редовище виконання [12]. Він має подібні функції, як веб-сервер Apache в LAMP (Linux, Apache, MySQL і PHP). Код програми складається з пакетів, які забезпечують функціональність програми. Приклад цих функціональних можливостей включає в себе вхідні дані OAuth, маршрутизацію запитів і реактивний інтерфейс користувача.

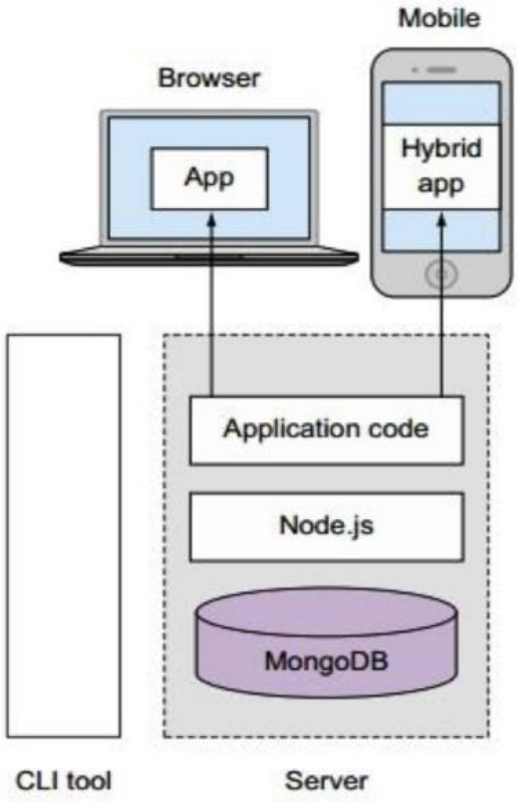

Рисунок 3.4 — Стек компонентів Meteor

Інструмент побудови, такий як CLI, допомагає у створенні та управлінні інфраструктурою середовища розробки, прискорюючи процес налаштування робочого процесу, який передбачає багато встановлення або налаштування за допомогою цих інструментів. У пакетах спільноти розробки Meteor, таких як npm і atmosphere, передбачено кілька інструментів, додавання аутентифікації через Gmail OAuth або Facebook OAuth можна швидко реалізувати за допомогою облікових записів користувачів Meteor до списку пакетів [13].

Браузер і мобільна платформа розгортають додаток і взаємодіють з користувачем. Таким чином, через розробку та тестування пакетів ядра у Meteor інтегровані пакети для роботи без будь-яких конфліктів. Незважаючи на використання специфічних пакетів з Meteor, також можна використовувати інші технології, що підтримуються Meteor.JS, наприклад, використання React.JS на стороні клієнта замість Blaze.JS [14].

3.4 Основні відомості про React

ReactJS, також відомий як React або React.js, є бібліотекою відкритих кодів JavaScript для створення інтерфейсів користувача. Він використовується для обробки шарів перегляду в додатках однієї сторінки та розробці мобільних додатків.

React прагне забезпечити швидкість, простоту і масштабованість. Деякі з найбільш помітних функцій - це JSX, компоненти зі станом, Virtual Document Object Model.

JavaScript XML (JSX) є розширенням синтаксису ECMAScript без будь-якої визначеної семантики [15]. React охоплює той факт, що логіка рендерінгу по суті пов'язана з іншою логікою інтерфейсу. Замість відокремлюючих технологій, React використовує вільно зв'язані одиниці, які називаються компонентами. JSX є необов'язковим і можна не використовувати в React. Однак, JSX - це хороша візуальна допомога при роботі з інтерфейсом користувача в JavaScript. Вона також дозволяє React показувати більш корисні повідомлення про помилки та попередження.

React дозволяє розділити інтерфейс користувача на незалежні, багаторазові частини, які називаються React компоненти. Компоненти реалізують метод візуалізації, який приймає вхідні дані і повертає те, що відображатиметься. Кожен компонент має кілька методів життєвого циклу, які можна перевизначити для виконання коду в певний час процесу.

State – це звичайний JavaScript-об'єкт, який використовується для запису та реагування на події користувача. Кожен визначений клас має свій власний об'єкт стану. Всякий раз, коли змінюється стан компонента, компонент і всі його дочірні компоненти негайно повторно відтворюють. Стани утримують значення по всьому компоненту і можуть передаватися до дочірніх компонентів як реквізит.

3.5 Огляд бази даних MongoDB

MongoDB – це документоорієнтована БД, що відноситься до нереляційних БД. Вона призначена для глибокої, масштабованої та швидкої роботи навіть для великих об’ємів даних. При проектуванні БД ще на початковому етапі закладається висока доступність, підтримка складних систем та просте розподілення даних по декількох серверах [16].

Нереляційні БД, або No SQL – це бази даних, що мають механізм видобування та зберігання даних не такий, як підхід таблиць-відношень у реляційних БД.

MongoDB зберігає дані у документах формату .json. Спеціальні запити, індексація та агрегація забезпечують потужні способи доступу до даних та їх аналізу в режимі реального часу [17].

Однією з ключових особливостей баз даних NoSQL і MongoDB, зокрема, є динамічні схеми [18]. Документи всередині однієї колекції можуть мати зовсім іншу структуру з полями різних типів. Схема задається не на рівні бази даних, а на рівні програми, що означає, що нові поля або документи з різною структурою можуть бути додані в разі потреби, навіть під час виконання. Розробникам не потрібно заздалегідь визначати схему. Ця функція особливо корисна для поліпшення або розширення програми, оскільки нові функції можуть бути легко додані без будь-якого простою і без часу, витраченого на перевизначення структури.

MongoDB має багато позитивних моментів, таких як проста і потужна JSON-подібна схема даних, досить гнучка мова запитів, динамічні запити, повна підтримка індексів, дуже швидкі оновлення, ефективне зберігання великих двійкових даних, журналювання операцій, які змінюють дані, підтримка відмови та масштабованості, а також MongoDB може працювати відповідно до парадигми MapReduce. Все це робить MongoDB дуже хорошим вибором для таких речей, як великі обсяги передачі даних, націлювання на оголошення, моніторинг соціальних медіа, великі обсяги метаданих, новини та різні програми управління контентом тощо.

3.6 Середовище розробки Visual Studio Code

В якості середовища розробки обрано Visual Studio Code, що надає весь необхідний функціонал для розробки веб-додатків.

Visual Studio включає в себе редактор коду, що підтримує технологію IntelliSense – компонент завершення коду. Інші вбудовані інструменти включають в себе: підтримку дебагінгу, підсвітку синтаксиса, сніпети та рефакторинг коду.

Підключення додаткової функціональності за допомогою плагінів дозволяє включити підтримку систем контролю версій, наборів інструментів для інших аспектів життєвого циклу розробки програмного забезпечення.

Вбудована підтримка таких мов програмування, як JavaScript, TypeScript і Node.js. Має багату екосистему розширень для інших мов, таких як C++, C#, Java, Python, PHP, Go, і режимів виконання .NET і Unity.

Visual Studio Code має декілька розширень для FTP, що дозволяє використовувати програмне забезпечення як безкоштовну альтернативу веброзробці. Тому код можна синхронізувати між редактором і сервером, не завантажуючи додаткового програмного забезпечення.

Visual Studio Code є редактором коду, який можна використовувати для різних мов програмування. Замість проектної системи користувачам дозволяється відкривати один або більше каталогів, які потім можуть бути збережені в робочих просторах для майбутнього повторного використання. Він підтримує ряд мов програмування і набір функцій, які відрізняються для кожної мови. Небажані файли та папки можна виключити з дерева проектів за допомогою налаштувань.

3.7 Вимоги до технічного забезпечення

Для роботи з програмою користувачу достатньо мати комп’ютер з будь-якими технічними характеристиками. Обов’язковою умовою є встановлений на ньому веб-браузер, крім Internet Explorer, та доступ в інтернет.

У разі, коли з програмним продуктом взаємодіє розробник, то до складу технічних засобів повинні входити:

– комп’ютер із наступною конфігурацією:

1) процесор з тактовою частотою не нижче 1.5 ГГц;
2) 32- або 64-розрядна операційна система;
3) достатній об’єм оперативної пам’яті (не менше 2 Гб);
4) інші складові можуть мати будь-які параметри, тому що вони не дуже впливають на роботу програми;

– додатково має бути встановлене таке програмне забезпечення:

1) база даних MongoDB;
2) Node.Js;
3) Sublime Text, але достатньо будь-якого текстового редактору;

– комп’ютерна периферія, до складу якої входить:

1) монітор;
2) мишка;
3) клавіатура.

У розділі технологічної бази розробленої системи було описано засоби розробки, що використовувалися для створення застосунку для дипломного проекту. При розробці програмного продукту було використано такі засоби розробки як HTML5, React, JS, Meteor та MongoDB.

4 РОЗРОБКА ЗАСТОСУНКУ

4.1 Архітектура сервісу

Веб-застосунок — це клієнт-серверний додаток, де браузер діє як клієнт і веб-сервер як сервер. Логіка веб-додатків розподіляється між клієнтом і сервером, зберігання даних виконується в основному на сервері. Дані обмінюються по мережі за допомогою протоколу передачі гіпертексту (HTTP). Однією з переваг такого підходу є те, що користувачі не залежать від конкретної операційної системи або конфігурації апаратних засобів. Таким чином, веб-додатки є крос-платформенними сервісами.

Розробка фронтенду — це процес створення публічної частини веб-сайту, яка безпосередньо контактує з користувачем.

Структура веб-сторінки, заснована на захоплених користувачем подіях, таких як клацання миші. Більш того, однією з найбільш корисних функцій в JS є асинхронний JavaScript і XML (AJAX), ця техніка може бути використана для надсилання та отримання від сервера всіх необхідних даних без оновлення веб-сторінки.

Розробка бекенду стосується реалізації серверної сторони, яка в першу чергу фокусується на логіці веб-додатків або, іншими словами, на тому як працює програма. Це процес створення ядра веб-додатків, розробка платформи для додатків і заповнення її всіма необхідними функціональними можливостями. Серверна сторона управляє даними, отриманими з інтерфейсу, і повертає результат назад у формі, зрозумілій для клієнта. Бекенд застосунку складається з трьох основних частин: програмного забезпечення веб-сервера, логіки програми та бази даних [19].

Програмне забезпечення веб-сервера — це програма, яка працює на апаратних засобах і обслуговує дані для клієнтів, які представлені у вигляді браузерів. Програмне забезпечення веб-серверу складається з декількох частин, але

ядро - це сервер HTTP [20]. Це програмне забезпечення знає, що таке Uniform Resource Locator (URL) і розуміє протокол HTTP.

Браузер відправляє запит, потім він досягає виділеного програмного забезпечення, а потім HTTP-сервер надсилає відповідь, що містить потрібну інформацію. При використанні протоколу HTTP завжди виконується певний набір правил. Перш за все, сервер здатний відповідати лише на запит, надісланий клієнтом. Він не може надсилати запити до веб-переглядача.

По-друге, сервер повинен надіслати відповідь на кожен вхідний запит, принаймні відповідь, що містить повідомлення про помилку. Клієнт отримає повідомлення про таймаут, що вказує на те, що сервер не відповів протягом заданого часу. Нарешті, кожен HTTP-запит повинен містити URL-адресу, що вказує конкретний сервер і шлях, на який цей запит повинен бути надісланий.

Логіка програми також називається бізнес-логікою сервера. Вона включає в себе всі операції по обробці запитаних і відправлених даних, збереження даних в базі даних, прийняття рішень про те, які дані потрібні, а потім запит необхідних даних з бази даних. Це дозволяє використовувати заходи безпеки, такі як створення механізмів аутентифікації для ідентифікації користувача, який запитує або передає дані від і до сервера. Авторизація є ще одним з заходів безпеки, які будуть впроваджені в логіці застосунку. Вона використовується для перевірки того, чи має цей користувач необхідні права на виконання потрібної операції, наприклад для застосунку віддалених замовлень це, власне, створення замовлення та його оплата.

Бізнес-логіка для застосунку створюється програмно з використанням різних мови програмування JS із використанням серверного програмного забезпечення Node.js. В даний час це майже завжди робиться за допомогою серверних фреймворків. Фреймворки веб-додатків полегшують розробку, охоплюючи і надаючи деякі основні функціональні можливості, такі як керування сеансами, механізми аутентифікації, форматування виводу, взаємодія з базою

даних. Серверний фреймворк для, який використаний в даному дипломному проекті — Meteor.Js.

Таким чином, ґрунтуючись на вищезазначеному, логіка застосунку є однією з найважливіших складових розробки, яка передбачає способи відображення, створення, збереження та маніпулювання даними. Безсумнівну важливість і загальне положення бізнес-логіки в розробці додатків продемонстровано на рисунку 4.2.

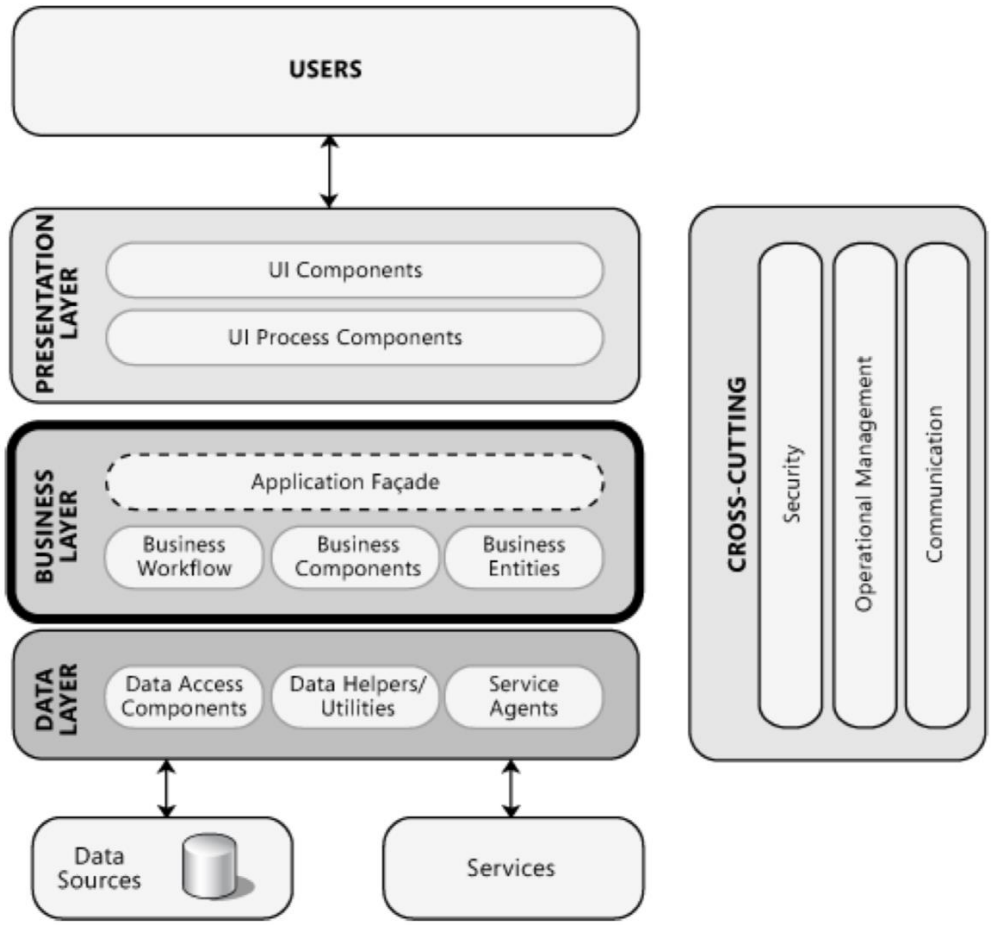

Рисунок 4.2 — Типова структура застосунку

На основі рисунку 4.2 видно, що бізнес-логіка займає одну з найважливіших частин у структурі програми. Реалізація всіх необхідних правил, обмежень і послідовностей процесів здійснюється шляхом кодування ділових правил реального світу в код, зрозумілий для машин. Розробники створюють спе-

ціальні робочі процеси як частину логіки застосунку, щоб визначити, як різні бізнес-об'єкти повинні взаємодіяти один з одним для досягнення необхідного результату.

Система управління базами даних (СУБД) відіграє надзвичайно важливу роль у розробці веб-додатків в цілому і як частина зворотного зв'язку зокрема. СУБД дає можливість збереження, модифікації та видалення даних. Це дозволяє запитувати (отримувати) дані з фактичної бази даних для подальшої обробки логічним шаром програми. СУБД надає способи визначення способу організації даних, запровадження різних заходів безпеки, збереження цілісності даних, створення і контролю правил паралелізму, додавання і моніторингу користувачів.

Бази даних NoSQL були створені у відповідь на обмеження традиційної технології реляційних баз даних. У порівнянні з реляційними базами даних бази даних NoSQL є більш масштабованими і забезпечують чудову продуктивність, а їхня модель даних вирішує кілька недоліків реляційної моделі. Нереляційні бази даних зазвичай не використовують SQL як мову для управління даними. Типи баз даних NoSQL глибоко відрізняються від звичайних систем управління реляційними базами даних. Порівнюючи табличне зберігання даних, що використовується RDBMS, бази даних NoSQL зберігають дані в різних форматах, таких як пари ключ / значення, графіки, широкі колонки, формат JSON або документ і т.д. Дані можуть бути дуже динамічними, а схеми зберігання повністю гнучкими. Як перевага такої гнучкості, кожен запис може мати різні властивості, а нові властивості можуть бути легко додані в разі необхідності без необхідності змінювати схему, що використовується базою даних, тому що схема диктується не базою даних, а програмою.

До переваг NoSQL можна віднести:

– Великі обсяги структурованих, напівструктурованих і неструктурованих даних;

– Об'єктно-орієнтоване програмування, яке є простим у використанні та гнучким;

– Ефективна, масштабна архітектура замість дорогої, монолітної архітектури.

Через вищезазначені переваги, для даного дипломного проекту було прийнято рішення використовувати нереляційну БД MongoDB. Для того, щоб зробити правильну структуру, що підходить до даного проекту було використано скрипт, написаний мовою JavaScript, завдяки якому було розбито всі файли з розширенням .json та внесено їх до БД у правильному порядку.

4.2 Підходи до розробки архітектури веб-застосунку

Історично склалося багато шаблонів MVC, які використовуються в різних технологіях, включаючи ASP.NET, PHP, Ruby on Rails, різні фреймворки JavaScript, а також бібліотеки. Хоча це і популярна архітектура для веб-додатків, існує також кілька товстих клієнтських технологій, таких як WPF, які є варіацією MVC паттерна.

Flux - нова архітектура додатків, розроблена Facebook для подолання обмежень архітектури MVC.

MVC - популярна і широко використовувана архітектура для розробки веб-додатків. Програми на базі MVC охоплюють різні сфери, такі як бізнес, магазини, освіта, уряд, охорона здоров’я. У цьому контексті справедливо припустити, що архітектура MVC є де-факто архітектурою, що використовується в сучасній розробці веб-додатків, незалежно від технології.

MVC базується на принципі SoC (Розділення проблем). Це означає, що кожна проблема (або відповідальність) відокремлюється таким чином, що кожен компонент вирішує конкретну проблему модульно.

Двонаправлений або односпрямований потік даних: сам шаблон дизайну MVC не ставить жодних обмежень, коли йдеться про потік даних між Model, View і Controller.

У MVC модель відповідає за керування та підтримку стану даних, а контролер виконує завдання оновлень даних на основі взаємодії з користувачем у представленні. Таким чином, контролер підтримує стан програми, а також є посередником між моделлю і представленням.

Отже, двостороння прив'язка даних добре працює в тих випадках, коли потрібно швидше оновлювати модель і представлення на основі взаємодії з користувачем. Насправді, двостороння прив'язка даних широко використовується в багатьох фреймворках JavaScript, включаючи Angular. Отже, ніхто не стверджує, що двосторонній потік даних в MVC є проблемою, не робить MVC помилковим шаблоном.

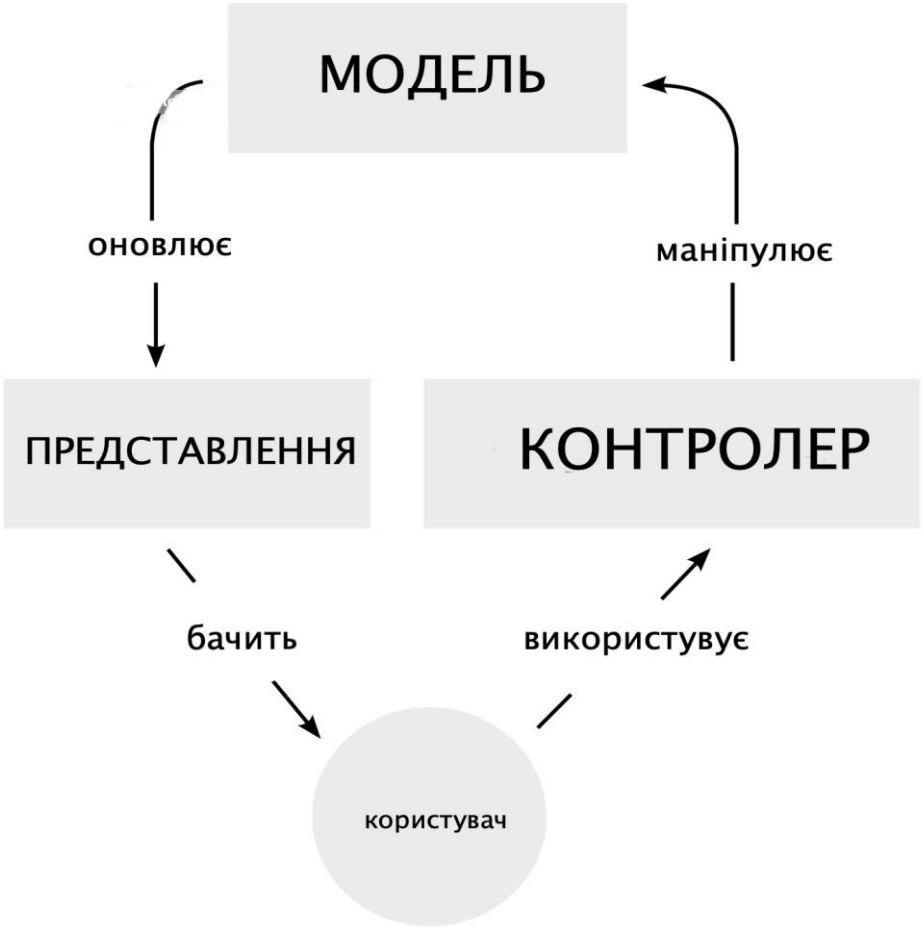

Рисунок 4.2 — Діаграма взаємодій у структурі MVC

Flux, як архітектура або зразок, має вбудовану структуру, яка передбачає односпрямований потік. Аргументом на користь односпрямованого потоку даних є те, що вона сприяє архітектурі чистого потоку даних. Це гарантує, що потоки даних у додатку в одному напрямку дають кращий контроль над ним. Він також сприяє послабленню зв'язку, оскільки стан програми міститься в конкретних сховищах.

MVC створює занадто багато контролерів та/або представлень: зазвичай будь-яка реалізація архітектури MVC має кілька контролерів.

Отже, якщо є складна програма, побудована за допомогою MVC, очевидно, що може бути стільки переглядів і контролерів, скільки функцій є. Це може створити проблему, коли багато переглядів і контролери взаємодіють

один з одним. Тим не менш, шаблон не зобов'язує мати один контролер на одне представлення.

Можна логічно об'єднати деякі контролери і відповідно оптимізувати складність структури коду. MVC не поширює жодного поняття структури одна модель - одне представлення - один контроллер. Поки кожен компонент керує SoC відповідно до принципів дизайну, можна мати логічну колекцію контролерів, моделей і навіть представлень.

Часто згадується, що MVC-додаток має проблеми з масштабуванням. Хоча архітектура може бути частково відповідальна за обмеження, пов'язані з масштабуванням, це не єдиний фактор. Існує багато веб-додатків, які розроблені за допомогою архітектури MVC і довели, що вони масштабовані для задоволення потреб клієнтів. Інформація є найбільш складним аспектом програмного забезпечення, коли справа доходить до масштабування. Flux добре охоплює масштабування інформації, оскільки ставить інформацію над усім іншим.

Суть Flux як архітектури полягає в потоці інформації в додатку.

Архітектура Flux має такі компоненти:

1) Дія: піднімається за допомогою представлення, коли користувач взаємодіє з елементами керування інтерфейсом у представленні.

2) Відправник: зберігає контекст до сховища даних і передає дію з представлення до сховища. Відправник отримує дію з представлення та сповіщає сховище.

3) Сховище: зареєстровано у відправнику. Сховище містить дані. Воно отримує подію оновлення від відправника і відповідає на нього. Відповідь буде іншою подією.

4) Представлення: відреагує на подію зміни і внесе відповідні зміни.

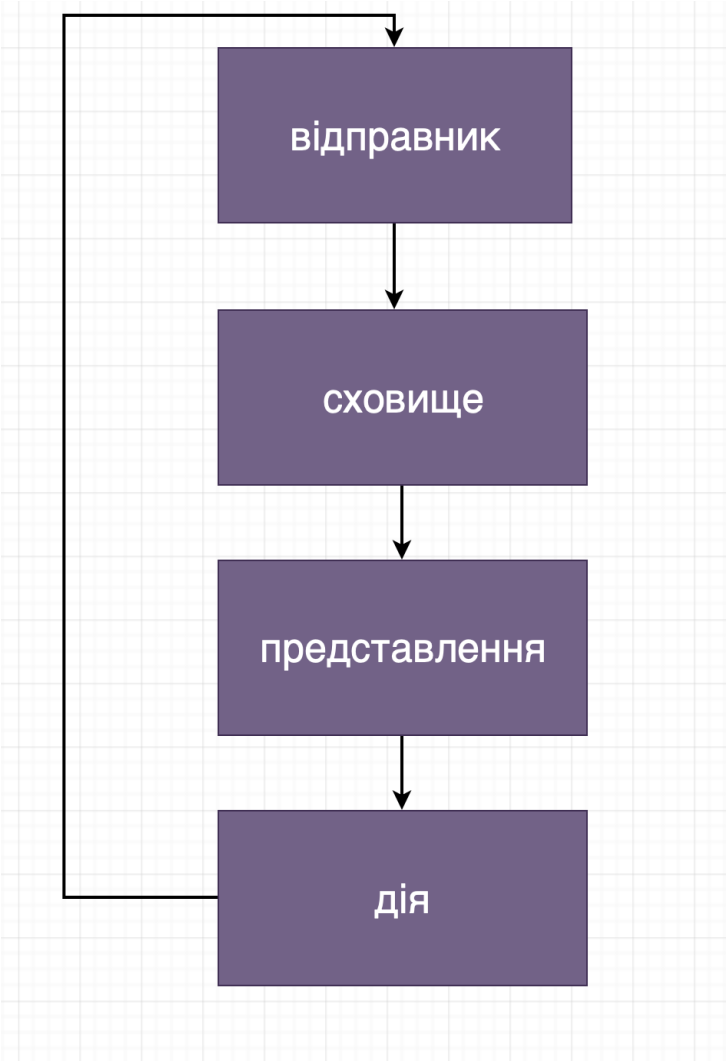

Рисунок 4.3 — Діаграма взаємодій у структурі FLUX

Redux є передбачуваним контейнером стану для додатків JavaScript. Redux - це спрощена реалізація архітектури Flux у Facebook, яка є структурою Model-View-Controller. За допомогою редукторів зменшується складність. Redux редуктори є функціями без побічних ефектів, які обчислюють наступний стан програми.

Redux заснований на трьох принципах:

1) Стан програми зберігається в одному об'єкті. Redux зберігає стан в одному об'єкті JavaScript, щоб полегшити відображення та передачу даних у всій програмі. Централізація стану в одному об'єкті також робить процес тестування і налагодження швидшим.

2) Стан програми є незмінним. У Redux, стани не можуть бути змінені. Єдиний спосіб змінити стан - це відправити дію. Дії є незмінними об'єктами JavaScript, які описують зміни стану.

3) Редуктори вказують, як дія перетворює стан. Редуктори - це функції JavaScript, які створюють новий стан з даним поточним станом і дією. Вони централізуються мутації даних і можуть діяти на всю або тільки частину стану. Редуктори також можуть бути об'єднані і повторно використані.

Ця архітектура значно збільшує масштабованість для великих і складних програм. Вона також дозволяє використовувати дуже потужні інструменти для розробників, оскільки можна простежити кожну мутацію до дії, яка її викликала. Зі станом і дією наступний стан програми можна передбачити з абсолютною впевненістю.

Коли користувач взаємодіє з елементом HTML, відповідна дія викликає відправку дії. Після цього відправник перевіряє, чи вимагає дія HTTP-запит до інтерфейсу API, використовуючи середню адресу. Якщо такі взаємодії необхідні, відправник повертає дію і чекає відповіді від бекенда, оскільки він асинхронний. Після того, як запит закінчений, дія потім передається назад. Потім відправник посилає дію і поточний стан на редуктор. Редуктор створює новий стан і замінює старий. Зберігання redux з оновленим станом повідомляє про будь-які компоненти перегляду, які змінюються станом. Потім компонент повторно відтворюється.

Використання архітектурного підходу Redux має такі переваги:

– Передбачувані оновлення стану полегшують розуміння того, як працює потік даних у додатку;

– Використання «чистих» редукторних функцій полегшує тестування логіки, а також дає можливість використовувати корисні функції, такі як «подорож у часі»;

– Централізація стану полегшує реалізацію подій, таких як внесення змін до даних або збереження даних між оновленнями сторінок

На додаток до цих загальних переваг, Redux забезпечує деякі переваги для збереження стану у застосуванні React.

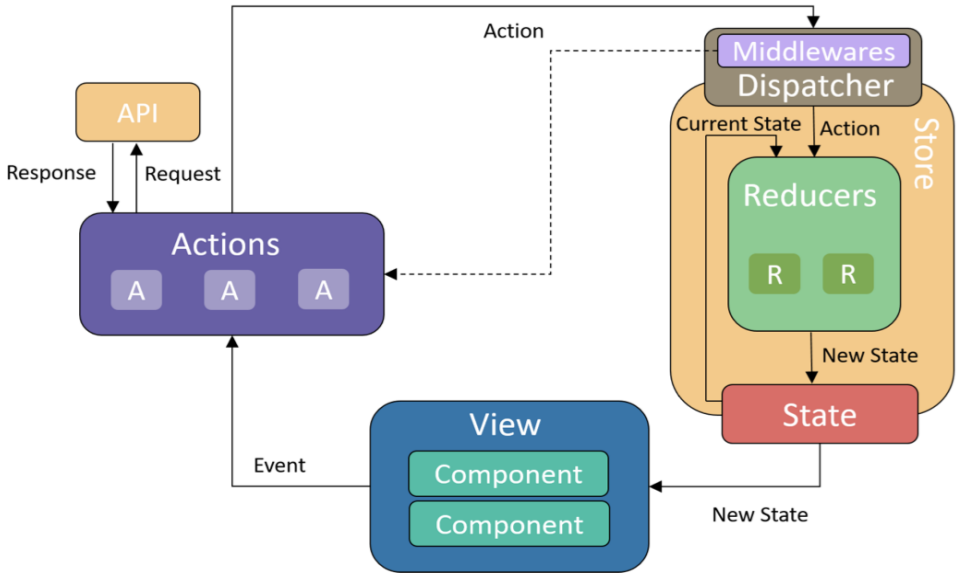

Рисунок 4.3 — Процес зміни стану системи з Redux

Отже, на основі вищезгаданих переваг, для розробки веб-застосунку для віддалених замовлень в закладах харчування було обрано архітектурних підхід — Redux.

4.3 Розробка веб-застосунку

Розроблюваний застосунок — це веб-застосунок для віддалених замовлень в закладах харчування, який призначений для суттєвої економії часу на очікуванні замовлень за рахунок автоматизації процесів створення та обробки замовлень. Для розробки веб-застосунку була обрана мова програмування JavaScript та такі фреймворки як React для клієнтської частини і Meteor — для серверної. В якості архітектурного паттерну для розробки веб-застосунку було обрано Redux. Розроблена платформа забезпечує частину всіх необхідних функцій, проте до програми легко можна додати більше функціональних можли-

востей. На момент написання даної дипломної роботи були реалізовані наступні інтерфейси:

- Інтерфейс для авторизації та реєстрації в системі користувача;
- Інтерфейс користувача для перегляду закладів та страв;
- Інтерфейс користувача для формування замовлень;
- Інтерфейс офіціанта для перегляду та обробки замовлень;
- Інтерфейс адміністратора для перегляду транзакцій;
- Інтерфейс адміністратора для редагування меню.

Як видно з наведеного вище списку, застосунок складається з трьох основних частин: інтерфейсів адміністратора, офіціанта та користувача.

4.3.1 Модуль системи керування меню

Цей модуль представляє адміністратора з функціями описаними далі, які доступні з панелі адміністратора.

Додавання страви. Додавання страви до меню закладу є основною функцією адміністративного модулю. Інформація про страви, що додаються тут, буде відображатися на загальнодоступній сторінці веб-сайту у розділі «Заклади».

Коли адміністратор додає нову страву, обробник подій onSubmit() в компоненті-контейнері AddMenuItem відправляє об’єкт дії з типом ADD_MENU_ITEM . Редьюсер реагує на отриману дію і відповідно до її типу змінює стан додатку, в ході чого відбувається перерисовка інтерфейсу відповідно до оновленого стану.

Ще одна основна функція адміністратора — створення категорії. Створення категорії відбувається аналогічно до додавання нової позиції меню: в обробнику події onCreateCategory за допомогою методу dispatch(addCategory(input.value)) відправляється дія ADD_CATEGORY, яка потім обробляється редьюсером.

В адміністративному модулі також створені функції для редагування ціни та редагування страви, які створюють дії EDIT_PRICE та EDIT_MENU_ITEM відповідно.

Для взаємодії клієнтської частини з сервером використовується Meteor.call, в якому вказується назва потрібного методу та параметри. За допомогою Meteor.call на сервері викликається вказаний метод, який вносить зміни до бази даних. В Meteor є можливість напряму створювати запити до бази даних з клієнтської частини, проте така взаємодія є незахищеною.

Коли метод викликається на клієнті за допомогою Meteor.call, паралельно відбуваються дві речі:

– клієнт надсилає запит на сервер для запуску методу в захищеному середовищі, як і запит AJAX;

– моделювання методу виконується безпосередньо на клієнті, щоб спробувати передбачити результат виклику сервера, використовуючи доступну інформацію.

Якщо результат з сервера повертається і відповідає моделюванню на клієнті, все залишається як є. Якщо результат на сервері відрізняється від результату моделювання на клієнті, інтерфейс користувача виправляється, щоб відобразити фактичний стан сервера. Вихідний код серверних методів представлено у додатку А.1.

4.3.2 Модуль онлайн замовлень

Відображення інформації про їжу на загальнодоступній сторінці веб-сайту. Інформація про страви, додані в модулі системи керування меню, повинна бути відображена на загальнодоступній сторінці веб-сайту, на яку користувач може перейти в браузері за посиланням …/restaurants/:id. Коли клієнт відвідує відповідну адресу URL, в класі RestaurantMenuList викликається метод DisplayRestaurantMenu, в якому викликається метод doPost. Він перевіряє,

який параметр приймається запитом. На верхній панелі користувач має можливість вибрати категорію, таку як «основні страви» або «напої». За замовчуванням, одразу при завантаженні сторінки встановлений параметр «все». Якщо метод doPost приймає параметр «все», вся інформація про страви буде відображена на сторінці, яка відповідає за відображення меню закладу. Якщо користувач вибирає іншу категорію, то буде відображена інформація про ті страви, які належать відповідній категорії.

Замовлення страв. Замовлені клієнтами страви, будуть тимчасово зберігатися в кошику. Реалізація кошику для замовлень досягається за допомогою сесій у Meteor. Сесії дозволяють зберігати невеликі фрагменти даних, які:

– не зберігаються в базу даних;

– не будуть запам’ятовані при повторному відвідуванні веб-застосунку відвідувачем.

Сесія надає глобальний об'єкт на клієнті, який можна використовувати для зберігання довільного набору пар ключ-значення. Для того щоб використовувати сесії в Meteor потрібно встановити пакет «sessions».

Страви, обрані клієнтами, будуть додані до кошика для замовлень. Якщо в сесії немає кошика, в сесії буде створено новий кошик з назвою «buycart». Тим часом «buyCardID» цього кошика буде додано до файлу cookie, і браузер клієнта збереже файл cookie з «buyCartID». Коли клієнт відкриває новий браузер, сервер отримає запит з файлом cookie. Файл cookie використовується для вирішення проблеми спільного використання кошика в кількох веб-переглядачах, він переконається, що попередній кошик не буде втрачений, коли клієнт відкриє новий браузер.

При натисканні кнопки «додати» у компоненті DishComponent викликається метод onAddToCart, який зберігає назву та ціну в колекцію CartCollection. При натисканні на зображення кошику у верхній панелі застосунку, користувач переходить на сторінку кошику замовлень, яка відрбражає компонент CartComponent. Код компонентів представлення та компонентів

контейнерів для страви та кошику замовлень наведений у додатку А.2 і додатку А.3 відповідно.

Авторизація. Компонент авторизації забезпечує вхід користувача в застосунок з-під власного аккаунту з використанням його електронної пошти та пароля. Для реалізації авторизації в Meteor було використано accounts-base пакет. Також в Meteor є такий протокол, як DDP. DDP (Distributed Data Protocol) - це вбудований RPC протокол, який є стандартним способом вирішення найбільшої проблеми, що стоїть перед розробниками JavaScript на стороні клієнта: запит на серверну базу даних, відправлення результатів клієнту, а потім надсилання змін клієнту, коли щось змінюється в базі даних.

На додаток до концепцій завантаження даних і викликів методів, DDP має ще одну вбудовану функцію - ідею поля userId у з'єднанні. Це місце, де відслідковується стан авторизації, незалежно від того, який пакет інтерфейсу облікового запису або служби для входу використовується. Ця вбудована функція надеє доступ до this.userId всередині методів і публікацій, і можна отримати доступ до ідентифікатора користувача на клієнті.

Пакет accounts-base включає в себе колекцію користувачів зі стандартною схемою, яка доступна за допомогою Meteor.users, а також Meteor.userId() та Meteor.user(), які представляють стан реєстрації на клієнті.

За замовчуванням функція Accounts.createUser, надана пакетом accounts-password, дозволяє створювати обліковий запис з іменем користувача, електронною поштою або обома. Код валідації нового користувача наведений у додатку А.4. Для розроблюваного застосунку було вирішено реалізувати авторизацію тільки за електронною поштою користувача, оскільки це надає декілька переваг:

– Скидання пароля. Коли користувач натискає на посилання у своїй електронній пошті, він перенаправляються на сторінку, де можна ввести новий пароль для свого облікового запису;

– Реєстрація користувачів. Коли новий користувач створюється адміністратором, пароль не встановлюється. Коли користувач натискає на посилання у своїй електронній пошті, він переходять на сторінку, де може встановити новий пароль для свого облікового запису. Цей спосіб дуже схожий на скидання пароля. Цю особливість було використано для створення можливості реєстрації в системі адміністратором нових офіціантів;

– Підтвердження електронною поштою. Коли користувач натискає посилання у своїй електронній пошті, програма записує, що цей лист дійсно належить правильному користувачеві.

На сервері для відправлення електронних листів користувачам використовуються такі методи: Accounts.sendResetPasswordEmail, Accounts.sendEnrollmentEmail, Accounts.sendVerificationEmail.

Також у веб-застосунку було реалізовано можливість авторизації за допомогою Facebook. Для цього використано пакет accounts-facebook. Також у Meteor є пакети для авторизації за допомогою Google, GitHub, Twitter, але в розробленому застосунку було використано тільки один, найпопулярніший і найпоширеніший спосіб - Facebook авторизація. Так як для реалізації авторизації використовується пакет accounts-ui, то для реалізації Facebook авторизації потрібно лише встановити потрібний пакет.

У Meteor за замовчуванням для даних користувача є колекція MongoDB. Вона зберігається в базі даних під назвою users і доступна у коді через Meteor.users. Схема користувацького документа в цій колекції буде залежати від того, яку службу входу було використано для створення облікового запису. Приклад структури користувача, який створив свій обліковий запис із accounts-password зображено на рисунку 4.3.3:

Структура об’єкту користувача, який здійснив вхід в систему за допомогою Facebook зображено на рисунку 4.3.4.

```
{
  "_id": "DQnDpEag2kPevSdJY",
  "createdAt": "2015-12-10T22:34:17.610Z",
  "services": {
    "password": {
      "bcrypt": "XXX"
    },
    "resume": {
      "loginTokens": [
        {
          "when": "2015-12-10T22:34:17.615Z",
          "hashedToken": "XXX"
        }
      ]
    }
  },
  "emails": [
    {
      "address": "ada@lovelace.com",
      "verified": false
    }
  ]
}
```


Рисунок 4.3.3 — Структура користувача в колекції users після входу за допомогою електронної пошти


```
{
  "_id": "Ap85ac4r6Xe3paeAh",
  "createdAt": "2015-12-10T22:29:46.854Z",
  "services": {
    "facebook": {
      "accessToken": "XXX",
      "expiresAt": 1454970581716,
      "id": "XXX",
      "email": "ada@lovelace.com",
      "name": "Ada Lovelace",
      "first_name": "Ada",
      "last_name": "Lovelace",
      "link": "https://www.facebook.com/app_scoped_user_id/XXX/",
      "gender": "female",
      "locale": "en_US",
      "age_range": {
        "min": 21
      }
    },
    "resume": {
      "loginTokens": [
        {
          "when": "2015-12-10T22:29:46.858Z",
          "hashedToken": "XXX"
        }
      ]
    }
  },
  "profile": {
    "name": "Sashko Stubailo"
  }
}
```


Рисунок 4.3.4 — Структура користувача в колекції users після входу за допомогою Facebook

4.3.3 Модуль обробки замовлення

Панель обробки замовлення для офіціанта відображається в класі WaitersPanel. Після підтвердження та оплати користувачем замовлення у кошику замовлень, інформація додається в базу даних до колекції Orders. Оскільки Meteor базується на веб-сокетах замість HTTP, то нові замовлення одразу відображаються на панелі обробки замовлення офіціанта класом DisplayNewOrders. WebSockets дозволяють як серверу, так і клієнту в будь-який час відправляти повідомлення без будь-якого відношення до попереднього запиту.

В модулі обробки замовлень реалізовано такий функціонал, як підтвердження замовлення, та встановлення його статусу. Для встановлення статусу є такі функції:

– acceptOrder — встановлює для замовлення статус «в обробці»;

– setOrderPreparing(time) — встановлює для замовлення статус «приготування» та приймає як параметр час в хвилинах, який вказує орієнтовний час до кінця приготування замовлення. Після встановлення часу та стутусу, в обробнику події за допомогою методу dispatch, відправляється об’єкт дії з типом SET_TIMER, який запускає таймер зворотнього відліку на панелі користувача в розділі замовлень, а також на панелі обробки замовлень в розділі «Поточні замовлення». Код реалізації класу OrderTimer представлено у додатку А.5;

– setOrderReady — встановлює для замовлення статус «готовий». Це означає, що замовлення готове, але відвідувач ще не отримав його;

– completeOrder — встановлює для замовлення статус «завершений» після того, як відвідувач прийшов у заклад і отримав послугу.

4.4 Діаграма послідовності

Діаграма послідовності (sequence diagram) - UML-діаграма, яка представляє взаємодію між лініями життя як упорядковану послідовність подій. У графічному матеріалі представлена діаграма послідовності, що описує процес оформлення замовлення, згідно сформульованим вимогам до веб-застосунку. В даному процесі користувач додає в кошик страви, які він хоче включити в своє замовлення. Після додавання всіх необхідних страв, користувач натискає кнопку «Оформити замовлення»

Якщо в момент натискання кнопки «Оформити замовлення» кількість позицій в кошику менше однієї, застосунок повідомляє користувача, що оформити замовлення можна, тільки додавши в кошик щонайменше одну позицію. Якщо в кошику знаходиться достатня кількість позицій, здійснюється перехід на сторінку оформлення замовлення, де користувач вводить персональні дані і здійснює оплату.

Після введення користувачем даних відбувається їх перевірка на коректність: ПІБ має бути не пустим, e-mail і номер телефону повинні відповідати стандартному формату, а зазначений час має бути в інтервалі найближчих двох днів у період роботи магазину. Якщо хоча б одне поле містить некоректні дані, додаток повідомить користувача про необхідність виправити цю інформацію. Якщо всі введені дані є коректними, додаток відправляє інформацію про замовлення на сервер, отримує відповідь від сервера про успішність оформлення замовлення і показує відповідне повідомлення користувачу.

4.5 Діаграма розгортання

У графічному матеріалі наведено схему структурну розгортання.

Опишемо більш детально схему структурну розгортання. Програма складається із серверної, клієнтської частин та сервера бази даних.

Ієрархія наступна: Сервер бази даних, у який входить компонент бази даних MongoDB; Веб-сервер складається з двох компонентів: презентаційний рівень та інтерфейс бази даних; Робоча станція, у яку входить Веб-браузер.

4.6 Діаграма діяльності

У графічному матеріалі наведено діаграму діяльності.

Початковим станом системи є запуск веб-застосунку. При запуску програми користувачеві показується початкова домашня сторінка. Натиснувши на кнопку «знайти заклад», користувач переходить на сторінку каталогу закладів, де може вибрати потрібний. При натисканні на будь який заклад, користувач переходить на сторінку меню, де може додати страву у кошик. Система може перейти в кінцевий стан в будь який момент діяльності, якщо користувач закрив додаток.

Процес оформлення замовлення описаний діаграмою послідовності оформлення замовлення.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Мета тестування

Метою тестування є перевірка відповідності функцій веб-застосунку для віддалених замовлень вимогам технічного завдання.

5.2 Загальні положення

Тестуванням програмного забезпечення є процес технічного дослідження, що дозволяє оцінити якість програмного забезпечення відповідного до вимог та відносно контексту в якому він використовується, а також виявлення можливих помилок у роботі.

Тестування розділяється на статичне, що виконується як перевірка відповідності програми її специфікації та технічному завданню, та динамічне, що виконується у ході прогону програми з різними вхідними даними та аналізом відмов та збоїв.

Розроблюваний додаток було протестовано за допомогою функціонального та інтеграційного тестування.

5.3 Функціональне тестування

Функціональне тестування - це тестування програмного забезпечення з метою перевірки реалізації функціональних вимог, тобто здатності програмного забезпечення в певних умовах вирішувати задачі, потрібні користувачам. Функціональні вимоги визначають, що саме робить програмне забезпечення, які завдання воно вирішує.

Розглянемо основні функції системи та перевіримо їх відповідність вимогам технічного завдання.

Тест № 1. Коректне відображення головної сторінки.

Вхідні дані: -

Ціль: Перевірити коректність відображення головної сторінки. У верхній частині сторінки повинна відображатися кнопка з посилання на вхід до системи, посередині сторінки повинна відображатися кнопка з посиланням на сторінку зі списком закладів.

Результат: тест пройдено.

Тест № 2. Коректне відображення закладів вибраного міста.

Вхідні дані: Застосунок відкрито на загальній сторінці зі списком закладів. Місто вибране автоматично або користувачем, визначення геопозиції включено.

Ціль: Перевірити коректність відображення списку закладів, відсортованих по відстані до користувача, а також коректне відображення закладів на мапі.

Хід проведення:

1) відкрити сторінку зі списком закладів;
2) перевірити коректність сортування по відстані;
3) перейти на сторінку з мапою;
4) перевірити відображення закладів на мапі у вигляді маркерів;
5) натиснути на маркер та переконатися в коректності відображення інформації про заклад.

Результат: тест пройдено.

Тест № 3. Коректність вибору закладу.

Вхідні дані: застосунок відкрито на загальній сторінці зі списком закладів. Місто вибране автоматично або користувачем. Заклад не вибраний.

Ціль: Перевірити коректність відображення та оновлення страв в залежності від вибору закладу.

Хід проведення:

1) перейти на сторінку закладу;

2) Перевірити коректність відображення страв та загальної інформації про заклад;

3) повернутися на загальну сторінку закладів;

4) перейти на сторінку іншого закладу;

5) перевірити відповідність відображення списку страв та загальної інформації.

Результат: тест пройдено.

Тест № 4. Коректність додавання товару в кошик.

Вхідні дані: Застосунок відкрито на сторінці закладу зі списком страв.

Ціль: Перевірити коректність додавання страви в кошик замовлень.

Хід проведення:

1) натиснути «додати» пі описом страви;

2) перейти на сторінку кошику;

3) переконатися, що вибрана страва була додана в кошик;

Результат: тест пройдено.

Тест № 5. Неможливість оформлення замовлення, якщо в кошику пусто.

Вхідні дані: Застосунок відкрито на сторінці зі списком закладів.

Ціль: Перевірити неможливість оформлення замовлення, коли кошик замовлень порожній.

Хід проведення:

1) перейти на сторінку кошика замовлень;

2) переконатися, що в кошику не відображається жодна зі страв;

3) натиснути на кнопку «замовити»;

4) перевірити появу повідомлення про те, що страви не були додані в кошик.

Результат: тест пройдено.

Тест № 6. Можливість оформлення замовлення, якщо в кошику присутня хоча б одна позиція меню.

Вхідні дані: Застосунок відкрито на сторінці закладу.

Ціль: Перевірити можливість оформлення замовлення, якщо в кошику присутня страва.

Хід проведення:

1) додати страву натиснувши кнопку «додати»;
2) перейти на сторінку кошика замовлень;
3) перевірити, що загальна сума замовлення правильна;
4) натиснути на кнопку «замовити»;
5) перевірити, що застосунок відобразив сторінку оплати замовлення.

Результат: тест пройдено.

Тест № 7. Коректність роботи відгуків про заклад.

Вхідні дані: Застосунок відкрито на сторінці завершеного замовлення.

Ціль: Перевірити коректність роботи відгуків про заклад.

Хід проведення:

1) натиснути на кнопку «залишити відгук»
2) переконатися, що з’явилася сторінка з відгуками про заклад або сторінки з текстом «немає відгуків», якщо раніше ніхто не залишав відгук про заклад.
3) ввести некоректні дані;
4) натиснути на кнопку «надіслати»;
5) переконатися, що всі поля підсвічені червоним;
6) ввести коректні дані;
7) натиснути на кнопку «надіслати»;
8) переконатися, що застосунок відобразив сторінку з відгуками та повідомленням про те, що відгук успішно доданий;
9) переконатися, що поля для відгуку зникли;
10) переконатися, що відгук про заклад з’явився в списку.

Результат: тест пройдено.

Тест № 8. Можливість додавання нової категорії меню адміністратором.

Вхідні дані: Застосунок відкритий на сторінці керуючої панелі адміністратора.

Ціль: Переконатися в можливості додавання нової категорії адміністратором.

Хід проведення:

1) перейти на вкладку «категорії»;
2) натиснути на кнопку «додати категорію»;
3) ввести назву;
4) натиснути кнопку «додати»;
5) переконатися, що категорія з’явилася в списку;

Результат: тест пройдено.

Тест № 9. Можливість додавання нової позиції меню адміністратором.

Вхідні дані: Застосунок відкритий на сторінці керуючої панелі адміністратор.

Ціль: Переконатися в можливості додавання нової позиції меню адміністратором.

Хід проведення:

1) перейти на вкладку «меню»;
2) натиснути на кнопку «додати страву»;
3) заповнити потрібні поля;
4) натиснути кнопку «додати»;
5) переконатися, що страва з’явилася в списку страв;

Результат: тест пройдено.

Тест № 10. Можливість реєстрації користувача в системі

Вхідні дані: Застосунок відкрито на головній сторінці.

Ціль: Перевірити можливість реєстрації нового користувача в системі.

Хід проведення:

1) натиснути на кнопку «вхід»;
2) натиснути на посилання «створити новий аккаунт»;

3) ввести дані;
4) натиснути кнопку «зареєструватися»;
5) натиснути кнопку «вихід»;
6) переконатися, що було здійснено вихід користувача з системи;
7) натиснути кнопку «вхід»;
8) ввести дані, які були вказані при реєстрації;
9) натиснути кнопку «вхід»;
10) переконатися, що було здійснено вхід в систему.

Результат: тест пройдено.

5.4 Інтеграційне тестування

Інтеграційне тестування - повна перевірка програмного продукту після його зборки з метою виявлення помилок, що виникають в процесі інтеграції програмних модулів або компонентів. Зібраний проект був перевірений на всіх можливих розширеннях екранів, а саме: 320px (Mobile S), 375px (Mobile M), 425px (Mobile L), 768px (Tablet), 10246px (Laptop) і 1440px (Laptop L) на предмет некоректності відображення елементів інтерфейсу. Помилок не було виявлено, всі елементи на всіх розширеннях екрану відобразилися коректно.

Отже, випробування проводилися шляхом функціонального та інтеграційного тестування. Було описано основні функції системи та перевірено їх відповідність вимогам технічного завдання.

Для кожної функціональної вимоги було прописано окреме випробування, у якому заповненими полями є:

- початковий стан;
- вхідні дані;
- хід проведення:
- результат тесту.

Після першого проведення випробувань було виправлено всі виявлені

помилки та пройдено тестування повторно.

У результаті всі випробування пройдено успішно.

Також шляхом інтеграційного тестування застосунок був перевірений на коректність відображення елементів інтерфейсу на всіх можливих розширеннях екрану. В ході тестування всі елементи були відображені коректно.

ВИСНОВКИ

У результаті виконання дипломного проекту було розроблено веб-застосунок для віддалених замовлень в закладах харчування.

У розділі «Опису предметної області» було детально описано предметну область та функціональну модель. Специфікацію функціональної поведінки системи продано у вигляді діаграми варіантів використання. Було визначено, що безпосередньо із системою будуть взаємодіяти три актори: клієнт – це особа, яка може працювати із основними функціями системи; адміністратор – формує меню закладу та реєструє офіціантів; офіціант – обробляє замовлення. Відповідно до визначених варіантів використання було виявлено функціональні вимоги та встановлено їх пріоритетність. Перед початком роботи над дипломним проектом було здійснено пошук застосунків зі схожою функціональністю. Нині існують програмні продукти, які дозволяють створювати онлайн замовлення, але функціонал таких застосунків незначний.

У розділі програмного та технічного забезпечення було описано засоби розробки HTML5, CSS3, JS, Meteor та MongoDB, а також визначено вимоги до технічного забезпечення. В розділі розробки програмного продукту наведено діаграми послідовності, розгортання та активності, а також описано реалізацію трьох основних модулів застосунку.

У розділі з випробуваннями програмного продукту описано мету випробувань, загальні положення та наведено результати випробувань. Метою випробувань є перевірка відповідності функцій застосунку для онлайн замовлень вимогам технічного завдання. Випробування проводилися шляхом функціонального та інтеграційного тестування. Було описано основні функції системи та перевірено їх відповідність вимогам технічного завдання. Для кожної функціональної вимоги прописано окреме випробування.

Розроблений застосунок має перспективи подальшого розвитку. З урахуванням ускладнення бізнес-процесу і ростом вимог замовника, виникає потреба в розширенні функціоналу системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційні технології – засіб оптимізації діяльності підприємств [Електронний ресурс] / О. О. Байкарова, Л. М. Тарасюк // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2013. – № 11. – С. 177-182. – Режим доступу: http://nbuv.gov.ua/UJRN/

2. Allset [Електронний ресурс]. – Режим доступу: https://allsetnow.com

3. Skip [Електронний ресурс]. – Режим доступу: https://www.skip.com.au

4. Кантор И. Современный учебник Java Script / И. Кантор. – learn.javascript.ru, 2015. – 400 с.

5. JavaScript. TutorialsPoint official website [Електронний ресурс]. – Режим доступу: https://www.tutorialspoint.com/javascript.

6. ES6, ES8, ES2017: что такое ESMAScript и чем это отличается от JavaScript. [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/wtf-is-ecmascript/

7. Дронов В.А. HTML5, CSS3 и Web 2.0 /В. А. Дронов. – БХВ-Петербург, 2011. – 256 с.

8. Кириченко А.В., Хрусталев А.А HTML5+CSS3. Основы современного web-дизайна. – СПб.: «Наука и Техника, 2018. – 352 с.

9. Кириченко А.В., Хрусталев А.А HTML5+CSS3. Основы современного web-дизайна. – СПб.: «Наука и Техника, 2018. – 352 с.

10. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения / Б. Хоган. – 2-е издание. – СПб.: Питер, 2014. – 320 с.

11. Simon Holmes. Getting MEAN with Mongo, Express, Angular, and Node. Manning Publications, 2015. – 125 p.

12. Crockford JavaScript: The Good Parts. 1st ed. O'Reilly Media: O'Reilly — New York: Wiley, 1990. – 342 p.

13. Cinco de NodeJS — May's BayJax Celebrates Server-Side JavaScript with Ryan Dahl, Elijah Insua, and Dav Glass. – InfoWorld, 2009. – 274 p.

14. Krasimir Tsonev. Node.js By Example. Birmingham, United Kingdom: Packt Publishing Limited. – Biometrics, 1985. – PP. 23–34.

15. Codesido I. What is front-end development? Guardian official website [Електронний ресурс]. – Режим доступу: https://www.theguardian.com/

16. Seguin K. The Little MongoBD Book / K. Segium. – Attribution-NonCommercial 3.0 Unported, 2018. – 34 p.

17. Bass, L., Clements, P. & Kazman, Software Architecture in Practice. 2nd ed. Boston: Addison Wesley, 1999. – 288 p.

18. Поняття архітектури програмного забезпечення. [Електронний ресурс]. — Режим доступу: http://www.lib.mdpu.org.ua/e-book/web/Lec10.html

19. What is a web server? [Електронний ресурс] MDN official website – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server

20. NoSQL vs SQL [Електронний ресурс] MSDN official website – Режим доступу: https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs- sql

ДОДАТОК А

Лістинг компонентів додатку

А.1 Вихідний код admin/methods.js

```
import { Meteor } from 'meteor/meteor';
import { Mongo } from 'meteor/mongo';
import { check } from 'meteor/check';

export const Menu = new Mongo.Collection('menu');
export const Categories = new Mongo.Collection('categories');

Meteor.methods({
 'menu.insert'(name, text, price) {
  check(text, String);

  if (! this.userId) {
   throw new Meteor.Error('not-authorized');
  }

  Menu.insert({
   name,
   text,
   price,
   createdAt: new Date(),
   owner: this.userId,
  });
 },
 'menu.remove'(itemId) {
  check(itemId, String);

  Menu.remove(itemId);
 },
 'menu.update'(itemId, newname, newtext) {
  check(itemId, String);
  check(newname, String);
  check(newtext, String);

  Menu.update(itemId, { $set: { text: newtext, name: newname } });
 },
 'categories.insert'(text) {
  check(text, String);

  if (! this.userId) {
   throw new Meteor.Error('not-authorized');
  }
  Categories.insert({text});
```

```
  },
  'categories.remove'(itemId) {
    check(itemId, String);

    categories.remove(itemId);
  },
  'categories.update'(itemId, newtext) {
    check(itemId, String);
    check(newtext, String);

    categories.update(itemId, { $set: { text: newtext } });
  },
  'setprice'(itemId, newPrice) {
    check(itemId, String);
    check(newPrice, String);

    menu.update(itemId, { $set: { price: newPrice } });
  },
});
```

А.2 Вихідний код класу Dish

```
export default class Dish extends React.Component {
    render() {
     const { price, text, title, img, onAddToCart } = this.props;
      return (
     <div>
     <img src={img}></img>
     <span className="dishTitle">{title}</span>
     <span className="dishText">{tetx}</span>
     <span className="price"> {price}  </span>
        <button onClick={onAddToCart}>
          Add To Cart
       </button>
    </div>)
     }
  }
  Product.propTypes = {
    price: PropTypes.number,
    text: PropTypes.string,
    title: PropTypes.string,
    onAddToCart: PropTypes.func.isRequired
  }
```

А.3 Вихідний код класу CartItem

```
export default class CartItem extends React.Component {
    render() {
```

```
    const { title, price, onRemoveItem } = this.props;
     return (
    <div>
       <span className="dishTitle">{title}</span>
       <span className="price">Ціна: {price}</span>
       <button onClick={onRemoveItem}>
         Видалити
       </button>
    </div>)
    }
  }
  Product.propTypes = {
    price: PropTypes.number,
    title: PropTypes.string,
    onRemoveItem: PropTypes.func.isRequired
  }
```

А.4 Функція валідації нового користувача

```
Accounts.validateNewUser((user) => {
  new SimpleSchema({
    _id: { type: String },
    emails: { type: Array },
    'emails.$': { type: Object },
    'emails.$.address': { type: String },
    'emails.$.verified': { type: Boolean },
    createdAt: { type: Date },
    services: { type: Object, blackbox: true }
  }).validate(user);


  // Return true to allow user creation to proceed
  return true;
});
```

А.5 Вихідний код класу OrderTimer

```
import React from "react";
import ReactDOM from "react-dom";


import "./styles.css";

class TimerInput extends React.Component {
  render() {
```

```
            return (
              <div style={{marginLeft:100}}>
                <h3>Input your desired time</h3>
                <input type="number" value={this.props.value} onChange={this.props.handleChange}
required />
              </div>
            );
          }
        }

        class OrderTimer extends React.Component {
          render() {
            return (
              <div>
                <h1 style={{ fontSize: 100, marginLeft:100
}}>{this.props.value}:{this.props.seconds}</h1>
              </div>
            );
          }
        }

        class StartButton extends React.Component {
          render() {
            return (
              <div style={{ marginLeft: 130 }}>
                <button className="btn btn-lg btn-success" disabled={!this.props.value} on-
Click={this.props.startCountDown}>Start</button>
              </div>

            );
          }
        }

        class App extends React.Component {
          constructor(props) {
            super(props);
            this.state = {
              seconds: '00',
              value: '',
              isClicked : false
            }
            this.secondsRemaining;
            this.intervalHandle;
            this.handleChange = this.handleChange.bind(this);
            this.startCountDown = this.startCountDown.bind(this);
            this.tick = this.tick.bind(this);
          }

          handleChange(event) {
            this.setState({
              value: event.target.value
```

```
  })
}

tick() {
  var min = Math.floor(this.secondsRemaining / 60);
  var sec = this.secondsRemaining - (min * 60);

  this.setState({
    value: min,
    seconds: sec,
  })

  if (sec < 10) {
    this.setState({
      seconds: "0" + this.state.seconds,
    })

  }

  if (min < 10) {
    this.setState({
      value: "0" + min,
    })

  }

  if (min === 0 & sec === 0) {
    clearInterval(this.intervalHandle);
  }

  this.secondsRemaining--
}

startCountDown() {
  this.intervalHandle = setInterval(this.tick, 1000);
  let time = this.state.value;
  this.secondsRemaining = time * 60;
  this.setState({
    isClicked : true
  })
}

render() {
  const clicked = this.state.isClicked;
  if(clicked){
  return (
    <div>
      <div className="row">
        <div className="col-md-4"></div>
        <div className="col-md-4">
          <Timer value={this.state.value} seconds={this.state.seconds} />
```

```
          </div>
        </div>
      </div>
    );
    }else{
      return (
        <div>
          <div className="row">
            <div className="col-md-4"></div>
            <div className="col-md-4">
              <TimerInput value={this.state.value} handleChange={this.handleChange} />
              <Timer value={this.state.value} seconds={this.state.seconds} />
              <StartButton startCountDown={this.startCountDown} value={this.state.value} />
            </div>
          </div>
        </div>
      );
    }
  }
}
```